jamk.fi

Audiovirtojen joustava ohjaus verkkoympäristössä

Timo Saukonoja

Opinnäytetyö
Syyskuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), Ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Saukonoja, Timo	Opinnäytetyö, AMK	Elokuu 2017
	Sivumäärä 73	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Audiovirtojen joustava ohjaus verkkoympäristössä

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Ari Rantala

Toimeksiantaja(t)
Combitech Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa järjestelmä, joka pystyy autonomisesti määrittämään audiolähteen lähettämälle audiovirralle määränpään, ennalta määritettyjen sääntöjen mukaisesti. Tämän tavoitteen saavuttamiseksi tuli toteuttaa myös ohjausrajapinnat, joiden avulla eri sovellukset voivat kommunikoida toistensa ja työssä toteutetun sovelluksen kanssa.

Työssä toteutettiin Qt Creatoria käyttäen palvelinsovellus sekä tämän sovelluksen ja toteutuksen tueksi luotujen testisovellusten kommunikaation mahdollistavat ohjausrajapinnat. Ohjausrajapinnat toteutettiin Combitech Oy:n tarjoamaa Websocket-teknologian päälle rakennettua ETS-järjestelmää käyttäen. Palvelinsovellus käyttää ETS-järjestelmästä löytyviä komponentteja yhteyksien hallintaan ja viestien lähettämiseen sekä vastaanottamiseen. Kaikki sääntöihin ja sovelluksen autonomiaan liittyvä on kehitetty ilman valmiita komponentteja. Toteutetun palvelinsovelluksen lisäksi työssä käydään läpi palvelinsovellusta käyttävät Combitech Oy:n asiantuntijoiden toteuttamat sovellukset. Nämä kolme sovellusta ovat Qt- ja ETS-teknologioita käyttäen testikäyttöön toteutetut audiovirtaa lähettävä sovellus ja audiovirtaa vastaanottava sovellus sekä Node.js- ja Vis.js-teknologioita käyttäen toteutettu käyttöliittymä.

Työn lopputuloksena tuotettiin prototyyppinä toimiva kokonaisuus. Kokonaisuuteen kuuluvat sovellukset käyttävät toteutettuja ohjausrajapintoja tiedonvälitykseen. Palvelinsovellus voi ohjata testisovellusten toimintaa ohjausrajapintojen avulla. Palvelinsovellus voi kertoa siihen yhdistäneiden sovellusten tila- ja metatietoja käyttöliittymälle. Käyttöliittymän avulla voidaan ohjailla palvelinsovelluksen toimintaa. Palvelinsovellus osaa päätellä mihin vastaanottavaan sovellukseen sen tulee ohjata lähettävän sovelluksen tuottama audiovirta. Päättely tapahtuu vertaamalla palvelinsovelluksen tietoja siihen yhdistäneistä sovelluksista XML-tiedostossa määriteltyihin sääntöihin. Nämä säännöt ovat myös ohjelmoinnista tai tietotekniikasta tietämättömän käyttäjän ymmärrettävissä ja määriteltävissä.

Avainsanat (asiasanat)
C++, Qt, XML, API, Sääntökone, Tekoäly

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Saukonoja, Timo	Bachelor's thesis	August 2017
		Language of publication: Finnish
	73	Permission for web publication: x

Title of publication

Dynamic control of audio streams in network environment

Degree programme

Software Engineering

Supervisor(s)

Ari Rantala

Assigned by

Combitech Oy

Abstract

The objective of this thesis was to create a system that can autonomously define a destination for an audio stream sent by an audio source, using predefined rules. To accomplish this goal, a set of interfaces must be created to enable the communication between different applications that are part of said system.

A server application and a set of interfaces were designed and programmed using Qt Creator. The interfaces use a Websocket based system called ETS. ETS is created and maintained by Combitech Oy. The server application uses features from ETS to manage connections, send and receive messages. Everything related to the application's autonomy and rules was programmed without existing components. In addition to the programmed server application, the thesis covers the applications designed to be used as clients for said server application. These three applications are the application that sends a stream, the application that receives a stream, and the user interface. The first two applications were programmed using Qt- and ETS-technologies and the user interface was programmed using Node.js- and Vis.js technologies. These applications were programmed by professionals of Combitech Oy.

The result of the thesis is a prototype system consisting many application that can function as a whole. These application use created interfaces to communicate with each other. The server application can use the created interfaces to control the actions of the client applications. The server application can inform user interface clients with the state- and metadata of the connected applications. The user interface clients can control the actions of the server application. The server application can deduce to which receiving application the audio source should start sending the audio stream. These deductions are made by applying predefined rules in an XML file to the information of the connected applications held by the server application. These rules can be read and defined by users that do not possess a great deal of knowledge about programming or information technology in general.

Keywords/tags (subjects)

C++, Qt, XML, API, Rules engine, Artificial intelligence

Miscellaneous

Sisältö

Kuviot

Sanasto

API

Application Programming Interface eli ohjelmointirajapinta. Mahdollistaa eri ohjelmien välisen tiedonvaihdon.

Audioformaatti

Sisältää audiostreamin (ks. Audiostream) käsittelyyn tarvittavat parametrit (ks. Parametri).

Audiostream

Verkon yli lähetettävä digitaalista audiota (ks. Luku 3.11 Audio) sisältävä datavirta.

Debuggaus

Prosessi, jonka avulla ohjelmasta etsitään virheitä.

ETS

Combitech Oy:n monia toimintoja tarjoava ohjelmistokehys. Työssä sitä käytetään Websocket yhteyksien hallintaan ja viestien lähettämiseen tai vastaanoottamiseen.

Git

Ks. Luku 3.8 Versionhallinta.

Git repository

Ks. Luku 3.8 Versionhallinta.

Git master branch

Ks. Luku 3.8 Versionhallinta.

Manager

Sisällön hallintaan suunniteltu sovellus.

Parametri

Ohjelmalle tai funktiolle käynnistyksen yhteydessä välitettävä tieto.

Puskuri

Tiedon välityksessä käytettävä väliaikainen tietovarasto.

Päättelykone

Tekoälynä toimiva, käyttäjän puolesta asioita päättelevä komponentti.

Referenssi

Viite tai viittaus johonkin asiaan kuten tekstiin tai ohjelman koodiin.

Repository

Tietotekniikassa käytetty tiedon säilytyspaikka. Sisältää historian muutoksista sekä muutoksiin liittyvät viittaukset.

Source

Yleisnimitys audiolähteelle eli audiota *lähettävä* sovellus tai laite.

Sink

Yleisnimitys audiovastaanottimelle eli audiota *vastaanottava* sovellus tai laite.

Stream

Yleisnimitys audiovirralle eli Sourcelta (ks. Source) Sinkille (ks. Sink) liikkuva datavirta.

Sääntökone

Suorittaa tiettyjä toimintoja etukäteen määriteltyjen sääntöjen mukaan.

Solmu

XML-formaatissa tiedon merkityksen kertova osa: <Merkitys>Tieto</Merkitys>. Käyttöliittymässä kuvakkeen omaava kohta kuten host tai source.

Sprint

Ohjelmistokehityksessä käytetty termi ajanjaksosta, jonka aikana tietty määrä työtä on saatava valmiiksi.

Syntaksi

Kielen kielioppisäännöt eli tietylle kielelle ennalta määritellyt säännöt ja periaatteet, joilla se esittää tietoa.

UI

User interface. Ihmisen ja ohjelmiston välinen käyttöliittymä.

XML

Extensible Markup Language. Merkintäkieli, jonka tarkoitus on olla sekä ihmisen että tietokoneen luettavissa.

1 Työn lähtökohdat

1.1 Taustaa

Kriisitilanteiden aikana on erittäin tärkeää, että esimerkiksi maanpuolustuksesta vastaavat yksiköt pystyvät kommunikoimaan toistensa kanssa. On myös tärkeää saada tietoa vihollisen liikkeistä, jotta niihin voidaan reagoida oikealla tavalla.

Tämän kaltainen tarve on ollut olemassa siitä lähtien, kun ihminen asettui aloilleen pieniksi yhteisöiksi. ”Apua!”-huudolla on saatu viesti kylän toiselle laidalle asti esimerkiksi susilauman hyökätessä karjan kimppuun tai muinaisessa Kreikassa on voitu lähettää juoksija Ateenasta Spartaan pyytämään apua Marathonin taisteluun persialaisia vastaan.

Opinnäytetyön tekijän onneksi tietokoneaikana ei tarvitse huutaa tai juosta. Riittää, että käytössä on Websocket-protokollaa käyttävä yhteys erilaisten audiolähteiden ja audiovastaanottimien välillä.

Ennen tietokoneiden yleistymistä erilaisten datavirtojen ohjaamiseen käytettiin usein aiheeseen perehtynyttä asiantuntijaa. 50-luvulla isolla toimistolla saattoi olla oma puhelinkeskus, jossa useat tehtävään koulutetut asiantuntijat vastasivat puheluihin ja ohjasivat audiovirran saamansa tiedon perusteella eteenpäin oikealle henkilölle. Vaikka nykyään suurilla organisaatioilla saattaa olla asiantuntijoita vastaamassa vaihteessa tarvittaessa, on useissa tapauksissa puhelun ohjaus automatisoitu. Usein asiakaspalveluun soittaessa kuuluu ”Soitit X:n asiakaspalveluun, jos asiasi koskee asiaa Y paina yksi…”. Tässä tapauksessa järjestelmä pyytää soittajalta tiettyjä tietoja ohjatakseen puhelun oikeaan paikkaan ja täysin ilman ihmisen ohjausta. Käyttäjän painallukset kulkevat jonkinlaisen sääntökoneen läpi, joka palauttaa järjestelmälle soittajan tavoitteleman määränpään käyttäjän antamien painallusten osuessa ennalta määritettyihin sääntöihin. Ihminen pystyisi helposti samaan, mutta tietokone tekee tämän paljon halvemmalla ja väsymättä.

Puheluiden ohjauksen automatisointia on toteutettu vasta joitain vuosia, mutta tämän kaltaisia asiantuntijan päätöksentekokykyä simuloivia järjestelmiä rakennettiin suurissa määrin 80-luvulla ns. asiantuntijajärjestelmien (engl. Expert system) muodossa.

Kuten aiemmassa esimerkissä puhelinkeskuksesta, myös toimeksiantajan toimialalla on pitkään siirretty dataa analogisessa muodossa kaapeleita tai radioaaltoja pitkin. Analoginen tiedonsiirron vahvuuksia ovat viiveettömyys ja siirtyvän tiedon vastatessa suoraan lähetettyä tietoa, sitä ei tarvitse erikseen muuntaa vastaanottavassa päässä. Heikkouksia ovat muun muassa signaalin vaimeneminen siirrettävän matkan kasvaessa sekä kaapeleiden käytön kankeus. Esimerkiksi käyttötapaus, jossa audiosignaalia siirretään kymmeniä tai satoja kilometrejä analogisessa muodossa, on kallis ja monimutkainen toteuttaa. Saman käyttötapauksen voi toteuttaa halvemmalla ja helpommin, muuntamalla siirrettävä audiosignaali ensin digitaaliseen muotoon. Digitaalisessa tiedonsiirrossa signaali ei vääristy yhtä herkästi ja se on helpommin korjattavissa vääristymien sattuessa. Myös olemassa olevien tietoverkkojen käyttö helpottaa tiedonsiirtoa pitkillä matkoilla. Tavallista verkkojohtoa pitkin voi kulkea mitä tahansa tietoa digitaalisessa muodossa, kun taas analogisen audiosignaalin siirtoon tarkoitettua johtoa pitkin ei voi siirtää videosignaalia. Toimeksiantajan toimialalla on huomattu tarve tämän kaltaisille joustaville ja halvoille ratkaisuille.

1.2 Toimeksiantaja

Työn toimeksiantajana toimi Combitech Oy. Combitech Oy on turvallisuus- ja tietoturvaratkaisuja tarjoava yritys, joka työllistää Espoossa, Tampereella, Jyväskylässä ja Säkylässä noin 80 työntekijää. Combitech tarjoaa asiakkaillensa ohjelmisto- ja turvallisuusratkaisuja, turvallisuuskonsultointia, tietoliikenne- ja järjestelmäratkaisuja sekä järjestelmäintegraatioita. Combitech Oy on turvallisuuskonserni Saab AB:n omistama yritys (Combitech n.d.). Combitech Oy tekee läheistä yhteistyötä Suomen Puolustusvoimien kanssa ja yksi opinnäytetyön tavoitteista on herättää asiakkaiden, kuten Puolustusvoimien kiinnostus jatkokehitykseen.

2 Tehtävän kuvaus

2.1 Ongelma

Combitech Oy:llä on jo olemassa sovellus datavirtojen lähettämiseen lähteeltä vastaanottimelle. Uuden lähteen ottaessa yhteyden sovellukseen sille luodaan automaattisesti puskuri. Luotu puskuri on kolmannen osapuolen tarjoaman tallentimen sisäinen. Käyttäjä voi myös ohjata haluamansa datavirrat tiettyihin tallentimiin tai ulostuloihin (kuten tietokoneen kaiuttimet) taulukkomaisen käyttöliittymän kautta.

Nykyisen version ollessa toimiva kokonaisuus toimeksiantaja oli havainnut siitä puuttuvan tiettyjä toimintoja. Mahdollisen uuden version tuli olla nykyistä järjestelmää monipuolisempi ja älykkäämpi. Esimerkiksi jokaiselle lähteelle ei luotaisi automaattisesti puskuria, vaan se voitaisiin ohjata suoraan ulostuloon ilman tallennusta. Uutta versiota varten toimeksiantajalta puuttui tietoa vaihtoehdoista ja toteutustavoista datavirtojen ohjailun automatisoimiseksi sekä halutun toiminnallisuuden toteutuksen mahdolliseksi osoittava prototyyppi.

Ongelmaa tutkittiin täysin audiovirtojen näkökulmasta vaikka prototyypin lopullinen käyttötarkoitus oli ohjailla mitä tahansa datavirtaa, kuten tekstiä tai videota. Ongelman automatisointiin liittyvää osaa ei ollut tarkoitus ratkaista toteuttamalla liian monimutkaista järjestelmää, vaan tarkoituksena oli tutkia mahdollisuuksia ja toteuttaa kehittäjän näkökulmasta suhteellisen nopeasti toteutettava ratkaisu, joka täyttää käyttäjän tarpeet.

2.2 Tavoitteet

Opinnäytetyön ensisijaisena tavoitteena oli tutkia eri vaihtoehtoja audiovirtojen ohjailuun ja hallintaan. Tutkimistulosten pohjalta suunniteltiin ja toteuttiin prototyyppijärjestelmä, joka pystyy ohjaamaan audiovirtoja verkkoympäristössä ilman, että käyttäjän tarvitsee puuttua ohjauksiin. Ohjauksen tulee perustua sääntöihin, joita käyttäjä voi määritellä ajonaikaisesti ilman ohjelmointitietämystä tai konsultaatiota ohjelmis-

ton kehittäjältä. Mikäli aikaa jäisi, tarkoituksena oli toteuttaa myös miellyttävä ja helppokäyttöinen käyttöliittymä, jonka avulla käyttäjä pystyisi manipuloimaan automaattisesti luotuja audiovirtoja.

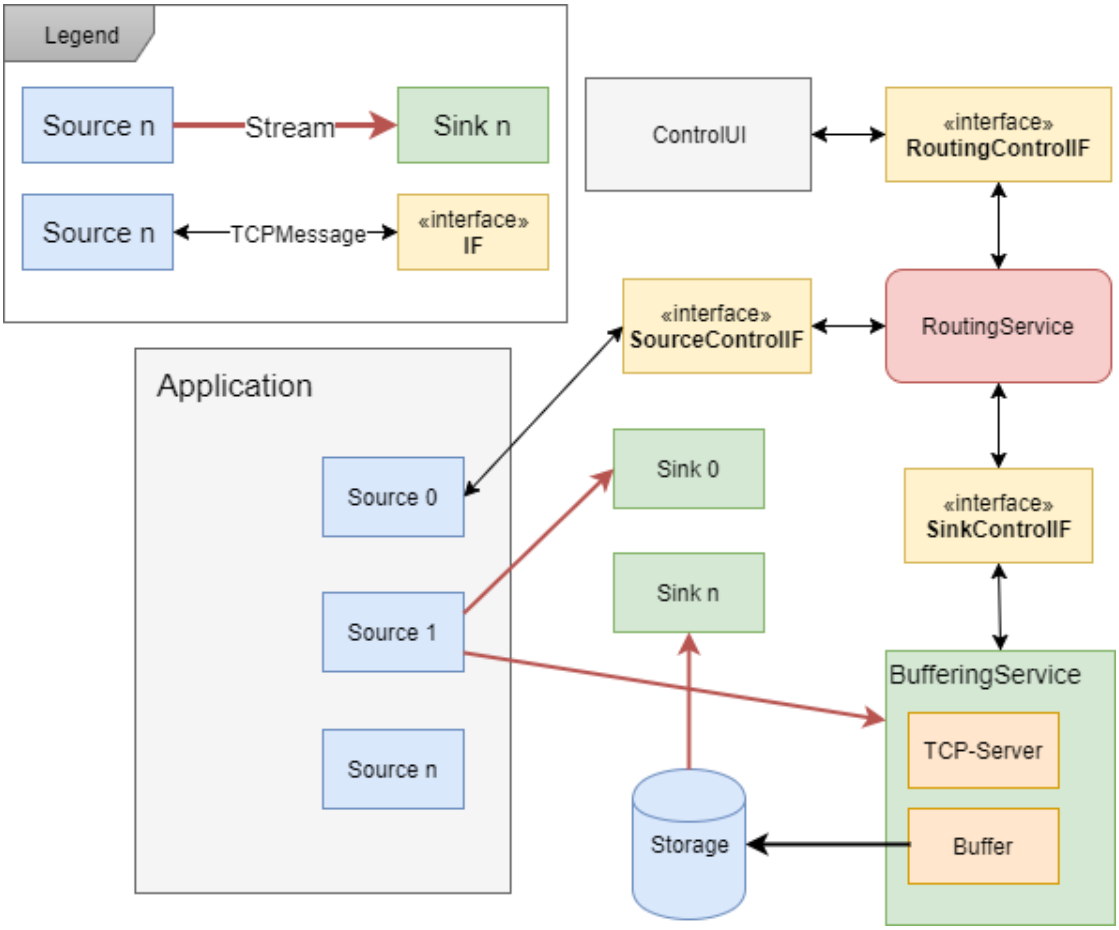
Toissijaisina tavoitteina oli perehtyä toimeksiantajan tapoihin ja toimintamalleihin, oppia toimimaan ja kommunikoimaan työyhteisössä sekä kehittyä projektityöskentelyssä ja projektiryhmän jäsenenä. Tämän lisäksi tarkoituksena oli kehittyä ohjelmoijana koodin ja dokumentaation luettavuuden ja ymmärrettävyyden osalta.

2.3 Vaatimusmäärittely

2.3.1 Yleistä

> *Vaatimusmäärittely kuvaa ohjelmistoprojektin tavoitteita ja vaatimuksia. Se määrittelee miten valmiin ohjelmiston tulisi toimia ja miten toiminnallisuus saavutetaan. Vaatimukset jaetaan yleensä toiminnallisiin ja ei-toiminnallisiin vaatimuksiin ja osa näistä vaatimuksista on järjestelmää koskevia rajoituksia* (Taina 2010.)

Kuvio 1 on vaatimusmäärittelyn perusteella projektin alkuvaihetta selkeyttämään piirretty kuvio, joka esittelee ohjelmiston eri komponentit. Ohjauksista vastaavan RoutingServicen lisäksi järjestelmästä löytyisi kolmen tyyppisiä sovelluksia: audiota lähettävät, audiota vastaanottavat ja näiden sovellusten väliset audiovirrat visualisoiva käyttöliittymä. Käyttöliittymän tarkoitus on myös hallinnoida sovelluksia ja niiden välisiä audiovirtoja. Suunnitteluvaiheessa piirrettyä kuviota 1 voi verrata samankaltaiseen toteutumaa kuvaavaan kuvioon 8.



Kuvio 1. Projektin alkuvaiheen kokonaiskuva ohjelmistosta

2.3.2 Kehitystyö

”Vaatimusmäärittely voi sisältää ei-toiminnallisia vaatimuksia siitä miten kehitystyötä tulee tehdä” (Taina 2010).

Projektin kehitystyötä koskevat vaatimukset eivät kuvaa itse toteutettavan järjestelmän toimintaa millään tavalla. Tästä syystä kaikki kehitystyötä koskevat vaatimukset vaatimusmäärittelyssä luokitellaan ei-toiminnallisiin vaatimuksiin. Käytännössä, ainakin tässä tapauksessa, kehitystyötä koskevat vaatimukset ovat rajoituksia.

Versionhallinta

Projektin versionhallintaan käytettiin Gittiä. Gitistä löytyvän repositoryn tuli olla yksityinen. Kaikki projektiin tuotettu koodi säilytettiin tässä repositoryssa tai projektiryhmän jäsenen henkilökohtaisella tietokoneella. Vain projektiryhmän jäsenillä tuli olla suora pääsy koodiin.

Repositoryssä sijaitsevasta master branchista löytyvän version tuli olla kääntyvässä kunnossa kaikilla ajan hetkillä. Mikäli näin ei syystä tai toisesta ollut, asia oli korjattava välittömästi.

Projektityöntekijä sai tarvittaessa luoda omia branchejä erilaisten ominaisuuksien testaamista varten.

Kehitystyö

Projektinhallintaan käytettiin JIRA-palvelua. JIRA mahdollisti kätevän työn jakamisen, jaksottamisen ja priorisoinnin eri projektiryhmän jäsenien kesken. Kehitystyö toteutettiin käyttäjälähtöisesti, sitoen jokainen suunniteltu ominaisuus käyttötapaukseen. Käyttötapaukset tuli jaotella järkevän kokoisiin tehtäviin JIRA:ssa työn jaksottamiseksi.

Ominaisuudet suunniteltiin ja toteutettiin ennalta sovittaviin sprintteihin sidotun aikataulun mukaan. Sprintti alkoi aina suunnittelupalaverilla, jossa suunniteltiin sprintin aikana toteutettavat ominaisuudet ja sprintti päättyi aina toteutetun toiminnallisuuden esittelyyn eli demoon.

Suunnitellut toiminnallisuudet toteutettiin käyttäen vain ilmaisia tai toimeksiantajan omistamia komponentteja.

2.3.3 Rajapinnat

Ongelman ratkaisulle oli olennaista, että audiovirtoja ohjaava sovellus pystyy vaihtamaan tietoja audiota lähettävien ja vastaanottavien sovellusten kanssa. Sovelluksien välille täytyi suunnitella ja toteuttaa rajapinnat, joiden avulla sovellukset voivat kertoa itsestään olennaisia tietoja audiovirran ohjausta varten ja vastaanottaa käskyjä audiovirran luomiseen tai poistamiseen.

Rajapintojen toiminnalliset vaatimukset

Rajapintojen tulee toteuttaa sovellusten väliseen kommunikointiin tarvittavat viestit. Viestien lähetykseen tulee käyttää Websocket-yhteyttä sovellusten välillä. Viestien muodostamiseen ja parsimiseen tulee käyttää toimeksiantajalta löytyvää ETS järjestelmää.

Rajapintojen ei-toiminnaliset vaatimukset

Viestien sisällön tulee olla sekä ihmisen, että tietokoneen luettavissa mahdollisten ongelmatilanteiden selvittämisen helpottamiseksi. Rajapinnat tulee dokumentoida tavalla, joka helpottaa ja nopeuttaa niiden käyttöä tulevaisuudessa. Rajapintojen toteutukseen tulee käyttää vain ilmaisia tai toimeksiantajan omistamia komponentteja.

2.3.4 Ohjauspalvelu

Ohjauspalvelun toiminnalliset vaatimukset

Ohjauspalvelun tulee voida lähettää ja vastaanottaa viestejä rajapintojen kautta. Ohjauspalvelun tulee pitää yllä tietopohjaa, joka sisältää tiedot kaikista siihen yhdistyneistä sovelluksista, sekä niiden välisistä audiovirroista. Ohjauspalvelun tulee voida luoda audiovirtoja sovellusten välille ennalta määritettävien sääntöjen mukaan. Ohjauspalvelun tulee lukea säännöt jokaisen päättelykoneen kutsun yhteydessä mahdollistaen ajonaikaisen määrityksen. Ohjauspalvelun tulee pystyä toimimaan ilman päättelykonetta eli käyttäjän tulee voida luoda audiovirtoja pelkästään käyttöliittymän avulla.

Ohjauspalvelun ei-toiminnalliset vaatimukset

Sääntöjen tulee olla helposti ihmisen ja tietokoneen ymmärrettävissä. Sääntöjen tulee olla luettavissa ja kirjoitettavissa myös teknologiaan vihkiytymättömiltä käyttäjiltä. Käyttäjän tulee voida määrittää uusia tai poistaa voimassa olevia sääntöjä sovelluksen ajon aikana. Ohjauspalvelu tulee toteuttaa käyttäen vain ilmaisia tai toimeksiantajan omistamia komponentteja. Audiovirrat eivät saa kulkea ohjauspalvelun kautta, vaan audiovirta tulee luoda suoraan lähettävästä sovelluksesta vastaanottavaan sovellukseen.

2.3.5 Käyttöliittymä

Käyttöliittymän toiminnalliset vaatimukset

Käyttöliittymän tulee käyttää rajapinnoissa määriteltyjä viestejä tiedon välittämiseen.

Käyttöliittymän ei-toiminnalliset vaatimukset

Käyttöliittymän tulee olla miellyttävän näköinen ja sulava käyttää. Käyttöliittymän tulee olla web-selaimella käytettävä. Käyttöliittymä tulee toteuttaa vain ilmaisista tai toimeksiantajan omistamista komponenteistä.

2.4 Työskentelyn ja tulosten muut rajoituksen

Projekti toteutettiin toimeksiantajan tiloissa, toimeksiantajan tarjoamilla laitteilla ja osa käytetyistä komponenteista on toimeksiantajan omistuksessa. Toimeksiantaja omistaa tuotetun koodin ja siihen pääsy rajoitetaan vain toimeksiantajan henkilöstöä koskevaksi. Opinnäytetyön tekijällä on oikeus käyttää itse tuottamaansa koodia referenssinä työn dokumentoinnissa arvioinnin mahdollistamiseksi. Toimeksiantajan omistamista valmiista komponenteista ei saa julkaista toimeksiantajan kriittiseksi määrittelemiä osia.

3 Teoriaa, teknologiat ja työkalut

3.1 Projektin vaihejako

Ohjelmistotuotannossa ohjelmistoprojektin kehitystyö jaetaan osiin onnistuneen tuotannon varmistamiseksi. Yleensä jaottelun tukena käytetään jotain valmiiksi kehitettyä ja julkisesti saatavilla olevaa vaihejakomallia (Ohjelmistotuotanto 2016). Projekti toteutettiin käyttäen ketteriin menetelmiin kuuluvaa Scrum-mallia.

Ketterät menetelmät puoltavat joustavaa suunnittelua, kehittyvää kehitystyötä, aikaista toimitusta ja jatkuvaa parantelua sekä rohkaisevat nopeaan ja joustavaan muutoksiin reagointiin. (Agile software development 2017.)

Scrum mallissa on vain kolme roolia: tuotteen omistaja (engl. product owner), Scrum-mestari (engl. Scrum-master) ja projektityöryhmä. Tuotteen omistaja toimii rajapintana asiakkaan suuntaan ja kerää kaikki vaatimukset tuotteen työlistalle(engl. backlog). Scrum-mestari vastaa sprintin tuloksesta ja toimii työryhmän tukena ongelmatilanteissa. Projektityöryhmä koostuu itse toteutuksen tekevistä ohjelmoijista, testaajista ja mahdollisesti dokumentoijista. Yleensä Scrum-mallissa työryhmä pyritään rakentamaan eri asioihin erikoistuneista henkilöistä kuitenkaan tekemättä ryhmästä riippuvaista yhdestä henkilöstä. Esimerkiksi henkilön A erikoistuessa käyttöliittymiin, henkilöiden B ja C tulee pystyä paikkaamaan henkilön A osaaminen hänen ollessa estynyt. Scrum-mallissa työryhmä on tarkoitus olla itseohjautuva, joten projektipäällikölle ei ole tarvetta. Usein kuitenkin Scrum-mestari ottaa projektipäällikön roolia muistuttavan roolin ja tämä saattaa olla tarpeellisista vähemmän kokemusta osaavien työryhmien kanssa. (Haikala & Mikkonen 2011, 48-49.)

Jokainen sprint alkaa sprint planningillä eli suunnittelukokouksella. Kokouksessa tuotteen omistaja esittelee backlogia, josta Scrum-mestari ja työryhmä yhdessä valitsevat ominaisuudet tai toiminnallisuudet, jotka on mahdollista toteuttaa sprintin aikana. Työlistalta valittavat alkiot jaetaan tehtäviin (engl. task), joiden kesto on yleensä 4-16 tuntia. Yksi tehtävä on yleensä yhden työryhmän jäsenen vastuulla, mutta se voi olla välttämätön myös jonkin toisen tehtävän toteutusta varten. Tehtävän saanut työryhmän jäsen voi jakaa sen edelleen ali-tehtäviin (engl. sub-task), jotka voivat helpottaa työn jäsennystä. Työn edistymistä seurataan Scrum-taulun (engl. Scrum-boardin) avulla, jossa tehtävät voivat sijaita kentissä To-Do (odottaa suorittamista), In Progress (työn alla) ja Done (valmis). Tarvittaessa tauluun määritetään lisää kenttiä esimerkiksi testausta varten. Sprintin päättyessä pidetään aina katselmointi (engl. sprint review). Työryhmä demonstroi eli esittelee sprintin aikana saavutetut tulokset ja kerää palautteen katselmointiin osallistuneilta. Katselmoinnin jälkeen pidetään Sprint retrospective, jossa arvioidaan sprintin toteutusta ja mietitään toimintatapojen kehittämistä. On syytä myös huomioida syntyneet haasteet ja ongelmat, jotta niihin voidaan ottaa kantaa seuraavan sprintin suunnittelukokouksessa. (Haikala & Mikkonen 2011, 49-51.)

”Viime vuosina yleisimmin käyttöön otettu ketterä menetelmä on **Scrum**, jopa niin että sanasta Scrum on tullut lähes sanan ketterä synonyymi” (Haikala & Mikkonen 2011, 46).

Scrum-menetelmän tehokkuus, suosio ja käyttöönoton helppous ovat luultavimmin syynä myös toimeksiantajan valinnalle tämän projektin toteutukseen.

3.2 Asiantuntijajärjestelmä

3.2.1 Taustaa

Asiantuntijajärjestelmä (engl. Expert system) on tietokonejärjestelmä joka pyrkii jäljittelemään asiantuntijan päätöksentekotapoja. Tämä tekee siitä eräänlaisen tekoälyn. Asiantuntijajärjestelmät on suunniteltu ratkaisemaan monimutkaisia ongelmia tulkitsemalla sille annettua tietoa, joka on yleisesti esitetty if-then muotoisina sääntöinä (jos *x* on totta, silloin *y* on totta tai suoritetaan *z*). (Expert system 2017.)

Ensimmäiset asiantuntijajärjestelmät kehitettiin 1970-luvulla Stanfordin yliopistolla Edward Feigenbaumin, jota joskus kutsutaan asiantuntijajärjestelmien isäksi johtaman projektin tuloksena. Asiantuntijajärjestelmiä pidetään ensimmäisinä oikeasti onnistuneina tekoälyinä. Asiantuntijajärjestelmiä alettiin käyttää laajasti 1980-luvulla ja kaksi kolmasosaa Fortune 500-listan yrityksistä käytti teknologiaa päivittäisessä toiminnassaan. 1990-luvulta eteenpäin termi expert system ja idea erillisestä tekoälynä toimivasta tietokonejärjestelmästä suurimmaksi osaksi hävisi IT-sanastosta. Syynä tälle uskotaan olevan asiantuntijajärjestelmien tarjoamien ominaisuuksien (kuten sääntökone) siirtyminen osaksi suurimpien sovelluskehitysyritysten (kuten SAP, Siebel ja Oracle) tarjoamia sovelluspaketteja. (Expert system 2017.)

Asiantuntijajärjestelmä koostuu kahdesta alijärjestelmästä: Päättelykoneesta (engl. Inference engine) ja tietovarastosta (engl. knowledge base) (Expert system 2017).

3.2.2 Päättelykone

Päättelykone on komponentti, joka päättelee uutta tietoa tietovaraston faktojen ja loogisten sääntöjen perusteella. Päättelykoneet toimivat yleensä yhdessä tai kahdessa eri tilassa: eteenpäin ketjuttavassa (engl. forward chaining) ja taaksepäin ketjuttavassa (engl. backward chaining). Eteenpäin ketjutettaessa päättelykone aloittaa tiedetyistä faktoista ja päättelee niiden avulla uusia faktoja. Taaksepäin ketjutettaessa päättelykone aloittaa päämäärästä ja määrittää tietovaraston avulla voiko annetun päämäärän saavuttaa tai pitääkö se paikkansa. (Inference engine 2017)

Esimerkiksi yksinkertainen sääntö ”Jos olet opinnäytetyöntekijä, olet opiskelija” esitettynä pseudokoodilla:

Sääntö1: Opinnäytetyön tekijä(x) => Opiskelija(x)

Eteenpäin ketjutettaessa päättelykoneen löytäessä Timo-nimisen olion, joka on opinnäytetyöntekijä, se päättelisi Timon olevan myös opiskelija

Taaksepäin ketjutettaessa päättelykoneelle voitaisiin antaa päämäärä: ”Onko Timo opiskelija?”. Tällöin päättelykone etsisi tietovarastostaan Timo-nimisiä opinnäytetyön tekijöitä, ja löytäessään sellaisen se voi todeta Timon olevan opiskelija.

Usein taaksepäin ketjutettaessa on tapana integroida käyttöliittymä päättelykoneeseen. Tällöin kun päättelykoneelta kysytään ”Onko Timo opiskelija?” ja päättelykone ei vielä tietäisi Timon olevan opinnäytetyön tekijä, se voisi kysyä käyttäjältä ”Onko Timo opinnäytetyön tekijä?” ja päätellä käyttäjän vastauksen perusteella, onko Timo opiskelija vai ei. Tämän kaltaisen käyttöliittymän integrointi päättelykoneeseen saattaa johtaa käyttäjän kannalta epäselviin tilanteisiin. Jos käyttäjältä yhtäkkiä kysytäänkin ”Onko Timo opinnäytetyön tekijä?”, hän ei välttämättä tiedä, miksi järjestelmä kysyy häneltä sitä. Tämä ratkaistaan generoimalla selitys käyttäjälle. Käyttäjän pyytäessä selitystä käyttöliittymän kysymykseen, hänelle voitaisiin kertoa ”Jotta voin määrittää, onko Timo opiskelija, minun on määritettävä, onko hän opinnäytetyön tekijä”. Tämän kaltaiset selitykset ovat olennainen osa käyttöliittymällistä päättelykonetta.

Ylempänä kuvattu Sääntö1 ei sinällään kuvaa vielä ”ketjua” kuten päättelykoneen tilat antavat ymmärtää, joten laajennetaan esimerkkiä hieman:

Sääntö1: Opinnäytetyön tekijä(x) => Opiskelija(x)

Sääntö2: Opiskelija(x) => saa opintotukea(x)

Jotta päättelykone voi päätellä, saako olio x opintotukea, sen on ensin pääteltävä, onko x opiskelija.

3.2.3 Tietopohja

Päättelykoneen tietopohja (engl. knowledge base) ovat kaikki faktat, jotka päättelykone tietää entuudestaan. Tietopohjan ja sääntöjen avulla päättelykone päättelee uusia faktoja tietopohjaan tai muuttaa olemassa olevien faktojen totuusarvoja. (Inference engine 2017.)

3.2.4 Säännöt

Yksi päättelykoneen komponenteista on loogiset säännöt, joiden avulla se voi päätellä uusia faktoja tai muuttaa olemassa olevien faktojen totuusarvoja. Sääntöjen tarkoitus on olla muidenkin kuin IT-osaajien luettavissa ja määriteltävissä. Kuviossa 2 on kuvattu yleinen säännön määrittelyyn käytetty syntaksi. Sääntö koostuu vasemmasta puolesta (engl. left hand side) ja oikeasta puolesta (engl. right hand side). Vasen puoli sisältää ehdot, joita päättelykone vertaa tietopohjaansa. Näiden ehtojen täyttyessä voidaan todeta oikealta puolelta löytyvä päämäärä todeksi. (Inference engine 2017)

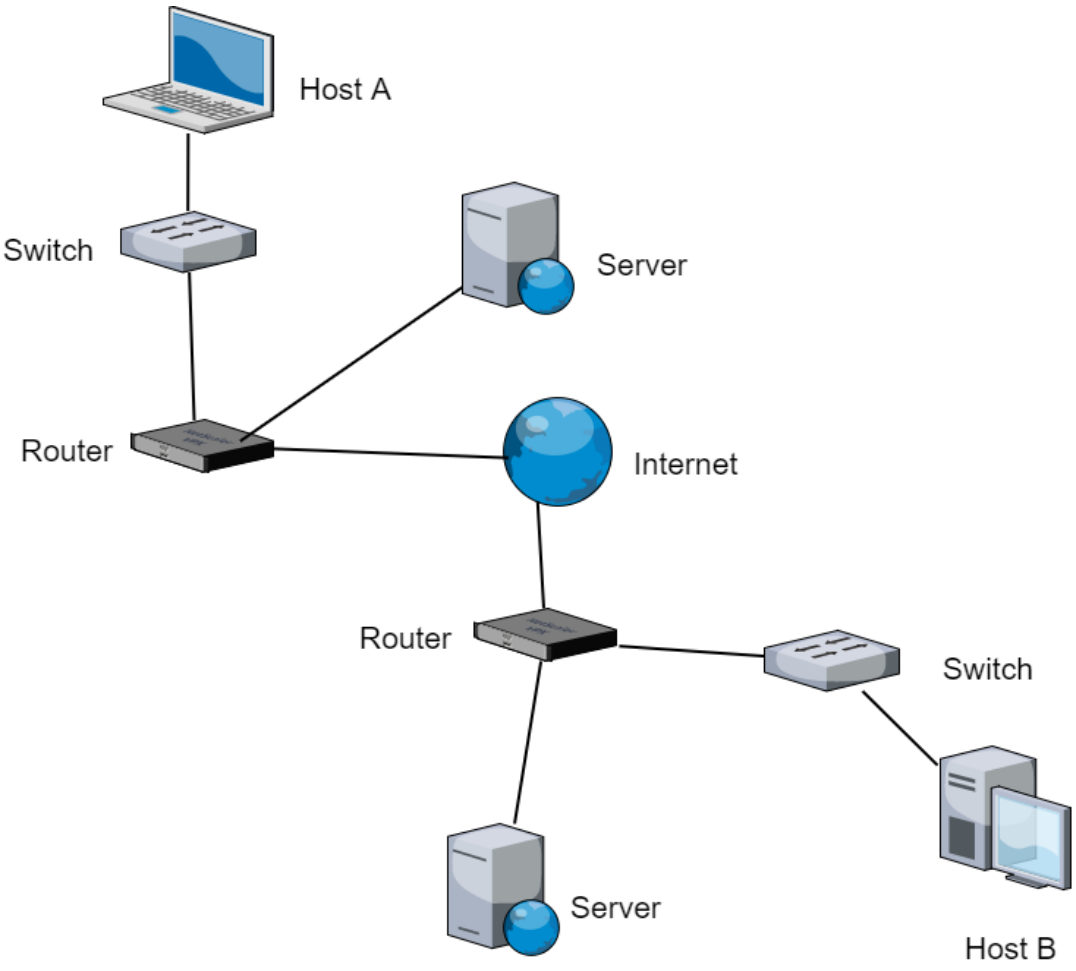
```
<Sääntö>
  <Ehto>Opinnäytetyön tekijä</Ehto> <!--Vasenpuoli -->
  <Ehto>Saa opintotukea</Ehto>      <!--Vasenpuoli -->

  <Päämäärä>Opiskelija</Päämäärä>   <!--Oikeapuoli -->
</Sääntö>
```

Kuvio 2. Esimerkki säännöstä esitettynä XML-formaatissa

3.3 Ohjauksen teoriaa

3.3.1 Yleistä

Opinnäytetyön ongelman tarpeena oli ohjata audiovirtoja lähettäviltä sovelluksilta vastaanottaville sovelluksille. Työssä toteutetun ohjauspalvelusovelluksen tehtävänä oli siis kertoa ohjauspalvelua käyttäville sovelluksille toistensa verkko-osoitteita, jotta ne pystyvät luomaan yhteyksiä välilleen audiovirran lähettämisen mahdollistamiseksi. Toimintona tämä tuo mieleen ohjelmallisen reitityksen, jota se ei kuitenkaan ole. Reitityksessä selvitetään tarkalleen vähiten resursseja kuluttava reitti usean verkon solmukohdan läpi. (Routing 2017.) Reitityksestä vastaava sovellus tietäisi jokaisen laitteen kuvion 3 mukaisessa verkossa. Työssä toteutettu ohjauspalvelu ei kuitenkaan ota kantaa verkossa kuljettavaan reittiin vaan ainoastaan siihen, luodaanko yhteys host A:n ja host B:n välille. Tästä syystä haluttiin välttää termin reititys käyttöä, kun puhutaan ohjauspalvelun suorittamasta toiminnosta. Sen sijaan käytetään termiä ohjaus.



Kuvio 3. Esimerkki verkon rakenteesta kahden koneen välillä

Käyttötapauksena tämä on niin ainutlaatuinen, että valmiita ratkaisuja ei ole saatavilla. Erilaisia avoimen lähdekoodin client-server-ratkaisuja mediavirran lähettämiseen laitteelta toiselle kyllä löytyy, mutta niitä yhdistävänä tekijänä on, että mediavirrat kulkevat palvelinsovelluksen kautta client-sovellukselle. Työssä ratkaistavan ongelman vaatimuksena on, että audiolähteet lähettävät audiovirrat suoraan audiovastaanottimille, jotta audiovirtoja ohjaava sovellus ei muodostu pullonkaulaksi aktiivisten audiovirtojen määrän kasvaessa. Tämän toteutus valmiilla palveluilla kuten Ampachella tai Icecastilla tarkoittaisi, että jokaisen audiolähteenä toimivan sovelluksen täytyisi olla oma audiovirran lähetyspalvelin. Tämänkaltainen ratkaisu on melko raskas pyörittää ja suhteellisen hankala toteuttaa, joten ainoaksi vaihtoehdoksi jäi toteuttaa oma ratkaisu.

3.3.2 Valmiita vaihtoehtoja

Ampache

Monipuolinen web-pohjainen median striimaussovellus. Ampachen avulla voi esimerkiksi kuunnella omia musiikkitiedostojaan millä tahansa webclientilla. Etuna Icecastiin mahdollisuus striimata pakkaamattomia audiotiedostoja. (Ampache n.d.)

Icecast

Ei yhtä monipuolinen kuin Ampache, mutta samaan tarkoitukseen suunniteltu sovellus. Icecastilla voi striimata vain pakattuja tiedostoja. (Icecast FAQ n.d.)

3.4 Ohjelmointirajapinta

Ohjelmointirajapintojen (engl. Application Programming Interface tai API) tarkoituksena on mahdollistaa sovelluksien välinen viestintä ja helpottaa ohjelmointia. Ohjelmoijan kehittäessä sovellusta, hän voi käyttää hyväkseen rajapintaa, joka tarjoaa ohjelmoijalle käyttöön hänen tarvitsemansa toiminnot ja oliot, mutta ei niiden takana olevaa toteutusta. Tämä yksinkertaistaa ja nopeuttaa ohjelmointia. (Application Programming Interface 2017.)

Esimerkkinä ohjelmoija kehittää sovellusta, joka visualisoi käyttäjälle säähän liittyvää dataa. Ilmatieteen laitos tarjoaa ohjelmoijalle käyttöön heidän ohjelmointirajapintansa, joka kertoo ohjelmoijalle, minkälainen pyyntö hänen sovelluksensa on lähetettävä, saadakseen vastauksena hänen kehittämänsä sovelluksen tarvitsema data. Ohjelmoijan tarvitsee opetella vain melko yksinkertaisen pyynnön lähettäminen ja datan parsiminen saadusta vastauksesta, mutta ohjelmoijan ei tarvitse tietää mitään ilmatieteen laitoksen tietokannasta tai sen kyselyistä.

Valmiita API ratkaisuja löytyy jonkin verran yleisimpiin käyttötapauksiin, kuten henkilötietojen hakuun tietokannasta. Projektin käyttötapaukset ovat niin uniikkeja, että ainoaksi vaihtoehdoksi jäi toteuttaa oma ratkaisu. Opinnäytetyössä toteutetut ohjelmointirajapinnat toteuttavat kahta toimintoa, tiedonvaihtoa ja ohjausta. Käytännössä kaikki ohjelmointirajapinnat toteuttavat tiedonvaihtoa jossain määrin, mutta sovellusten toiminnan ohjailuun suunnitellut rajapinnat eivät ole yleisiä. Tästä syystä opinnäytetyössä toteutetuista rajapinnoista on sopivaa käyttää termiä ohjausrajapinta.

3.5 Ohjelmointikieli

> *Ohjelmointikieli (Engl. Programming language) Tietokoneen ohjelmoinnissa käytetty kieli. Kielet voidaan jakaa kahteen ryhmään: kääntäviin ja tulkattaviin. Kääntäjä muuntaa ohjelmointikielellä kirjoitetun lähdekoodin prosessorin omalle kielelle. Käännös vie oman aikansa, mutta tuloksena oleva konekielinen ohjelma toimii nopeasti. Tulkki käsittelee lähdekoodia rivi tai käsky kerrallaan muuntaen sitä prosessorin ymmärtämään muotoon ajon aikana. Tulkkaus on hidasta, mutta hidas käännösvaihe jää kokonaan.* (Järvinen 2003, 471.)

Artikkelissaan Britton (2008) suosittelee keskittymään ohjelmointikieltä valittaessa seuraaviin kriteereihin:

- oppimisen helppous
- ymmärtämisen helppous
- kehitystyön nopeus
- apu oikeellisen koodin tuottamiseen
- ajonaikainen tehokkuus
- tuetut alustat
- siirrettävyys
- tarkoituksenmukaisuus.

Opinnäytetyön tekijän ohjelmointikielen valintaprosessi oli hieman yksinkertaisempi. Käytännössä valintaan vaikuttivat aiempi osaaminen, kiinnostus oppia uutta ja toimeksiantajan ammattilaisten suositukset. Opinnäytetyön tekijän ohjelmointikokemus keskittyy suurilta osin olio-ohjelmointiin, joten oli järkevää valita jokin olio-ohjelmoinnin mahdollistava kieli. Käytännössä vaihtoehdoiksi muodostuivat opinnäytetyön tekijän suhteellisen paljon käyttämä Java ja toimeksiantajan suosittelema C++, joka mahdollistaisi Qt-ympäristön käytön.

Olio-ohjelmointi on suosittu ohjelmointitekniikka, jonka tarkoituksen on vähentää ohjelmointivirheitä ja lisätä ohjelmointityön tuottavuutta. Olio-pohjainen koodi on usein myös uudelleen käytettävämpää kuin muilla tekniikoilla kehitetty koodi. (Järvinen, 2003, 474.)

Java on alustariippumaton olio-pohjainen ohjelmointikieli. Käännettyä Java-koodia voi ajaa millä tahansa Javaa tukevalla alustalla ilman tarvetta uudelleenkääntämiselle. Java perustuu C++:aan ja sen tarkoitus on olla helpommin omaksuttavissa. (Java (Programming Language) 2017.)

Qt on alustariippumaton käyttöliittymien ja ohjelmistojen kehitysympäristö. Qt laajentaa C++:aa käyttäen MOC-työkalua. C++ on suositun C-kielen laajennos. C++ suunniteltiin sisältämään parannuksia C-kieleen ja tukemaan abstrakteja tietotyyppejä ja olio-ohjelmointia. MOC on työkalu, joka käsittelee Qt:n C++-laajennokset tuottamalla niiden toimintaan vaadittua lähdekoodia. Qt-kehitysympäristö on erittäin pidetty ohjelmoijien keskuudessa sen monipuolisten ja helppokäyttöisten luokkakirjastojen vuoksi. (About Qt 2017; History of C++ n.d; Using the Meta-Object Compiler (moc) 2016.)

Valintaan vaikuttaneista kriteereistä Javaan liittyvä aiempi osaaminen teki siitä vahvan ehdokkaan. Kuitenkin toimeksiantajan edustajilta löytyvä vahva osaaminen C++:aan ja Qt-kehitysympäristöön sekä kiinnostus oppia uutta vahvistivat valinnaksi Qt:n.

3.6 Merkintäkieli

Merkintäkieli on tiedonesitystapa, jossa tiedon merkitys kuvataan tiedon seassa. Merkintäkielet ovat helposti sekä ihmisen että tietokoneen luettavissa. (Markup language 2017.)

Formaattivaihtoehdoiksi valikoituivat XML ja JSON näistä valmiiksi löytyvän kokemuksen perusteella. Qt:sta löytyvä QDomNode-luokka, joka on kirjaimellisesti luotu XML:n käsittelyä varten, kallisti valinnan XML:n suuntaan.

XML

Extensible Markup Language on merkintäkieli, jonka tarkoitus on olla sekä ihmisen, että tietokoneen luettavissa (XML 2017.)

JSON

JavaScript Object Notation on XML:n tapainen tiedostoformaatti, joka on helppolukuista ihmisille ja tietokoneille (Introducing JSON n.d.).

3.7 Käyttöliittymään liittyvät teknologiat

Käyttöliittymä toteutettiin osana ohjelmistoprojektia, mutta ei opinnäytetyöntekijän toimesta. Käyttöliittymä liittyy kuitenkin vahvasti opinnäytetyön tekijän toteuttamaan sovellukseen, joten teknologiat esitellään lyhyesti.

JavaScript

JavaScript on nykyään erittäin suosittu web-ympäristöissä käytettävä komentosarjakieli. JavaScript on HTML:n ja CSS:n rinnalla yksi kolmesta web-sisällön tuottamiseen käytetystä perusteknologiasta. (JavaScript 2017.)

Node.js

Node.js on suosittu palvelinpään JavaScript-toteutus (Node.js 2017).

Vis.js

Vis.js on visualisointikirjasto, joka mahdollistaa miellyttävän käyttöliittymän toteutuksen (vis.js 2017).

3.8 Versionhallinta

Versionhallinta on ohjelmistoprojektien yksi elinehto. Se mahdollistaa usean ohjelmoijan työskentelyn samojen järjestelmän osien kanssa yhtäaikaisesti ja helpottaa tehdyn työn yhdistämistä toimivaksi kokonaisuudeksi.

Git

Git on laajasti suosiossa oleva hajautettu avoimen lähdekoodin versionhallintajärjestelmä. Projekti toteutettiin pienessä projektiryhmässä ja Gittiä käytettiin versionhallintaan. Gitin toiminta perustuu repositoryihin, jotka toimivat tuotetun koodin säilytyspaikkana. Repository sisältää historian kaikista repositoryn koodiin tehdyistä muutoksista eli commiteista. Git mahdollistaa siirtymisen näiden committien välillä. (Getting Started – Git Basics n.d.)

Git erottuu muista versionhallintajärjestelmistä branching toimintonsa ansiosta. Branching toiminto mahdollistaa usean eri toiminnon samanaikaisen kehityksen. Jokaisesta repositorysta löytyy ensisijainen branch eli master branch. Master branch sisältää toimivan version koodista ja siitä löytyvään versioon tehdään vain täysin toimivia committeja. Lisättävien ominaisuuksien kehitystyö tapahtuu vaihtoehtoisissa brancheissa, joita Gitin käyttäjä voi itse määrittää. Tämä mahdollistaa versionhallinnan tehokkaan käytön usean ohjelmoijan kehittäessä ominaisuuksia samaan repositoryyn. (Git Branching n.d.)

Gitlab

Gitlab on kasvavaa suosiota saavuttava web-pohjainen repository manager Gitille. Gitlabin avulla Gitin käyttäjät voivat mm. tarkastella eri committien välisiä eroja, hallinoida repositoryn käyttäjien oikeuksia ja hallinnoida branchejä. Tämän lisäksi Gitlab tarjoaa mm. wikin, jossa voi säilyttää repositoryn sisältöön liittyvää dokumentaatiota ja issueiden, kuten bugien seurantaan liittyvää toiminnallisuutta. (Gitlab 2017.)

3.9 Projektinhallintatyökalut

Projektiryhmässä toteutettavan projektin hallinnan helpottamiseksi ohjelmistokehityksessä on tapana käyttää jonkinlaista työkalua. Useimmissa työkaluissa projektin toteutus jaetaan pienempiin kokonaisuuksiin ketterien ohjelmistokehitysmenetelmien mukaisten sprinttien avulla. Yleensä käyttäjätarinat kootaan backlogille, josta ne ripotellaan sprintteihin projektiin osallistuvien arvioiman työkuorman perusteella.

Redmine

Redmine on avoimeen lähdekoodiin perustuva web-pohjainen ja alustariippumaton projektinhallintatyökalu. Redmine on avoimen lähdekoodin toteutus, joten käyttö on ilmaista. (Redmine n.d.)

Jira

Jira on Atlassian nimisen yrityksen tarjoama web-pohjainen projektinhallintatyökalu. Jira on laajasti käytetty, mutta maksullinen työkalu. (Jira (software) 2017.)

Toimeksiantajan projekteissa ollaan siirtymässä Redminen käytöstä Jiran käyttöön, joten projektinhallintaan käytettiin Jiraa.

3.10 Tietoliikenne

Tietoliikenne on digitaalisen tiedon siirtämistä pisteestä pisteeseen tai pisteestä useampaan pisteeseen. Tiedon siirtäminen voi tapahtua esimerkiksi kuparijohtimien, optisten johtimien tai langattomien teknologioiden kuten radioaaltojen avulla. (Data transmission 2017.)

Erittäin suuri osa ihmisten päivittäin käyttämistä sovelluksista, kuten puhelimet, televisiot ja internet kaikki käyttävät tietoliikennettä muodossa tai toisessa.

Internet

Internet on verkkojen verkko eli maailmanlaajuisesti eri tietokoneverkkoja ja näin laitteita yhdistävä järjestelmä. Internet koostuu suurista määristä julkisia, yksityisiä, akateemisia ja liiketoiminnallisista verkoista joita yhdistää laaja valikoima erilaisia elektronisia, langattomia ja optisia teknologioita. (Internet 2017.)

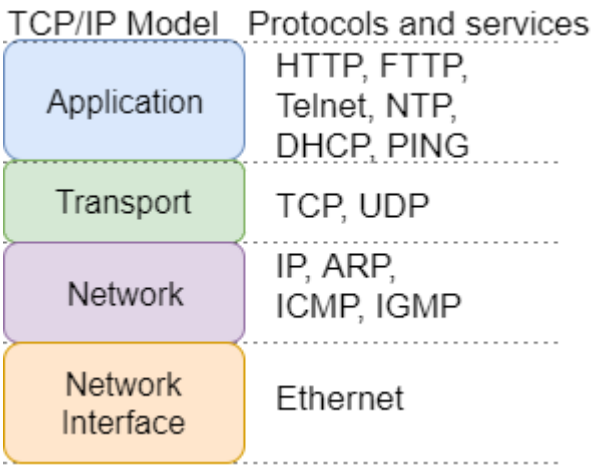
Viestintäprotokolla

Viestintäprotokolla on säännöistä muodostuva järjestelmä, joka mahdollistaa kahden tai useamman kokonaisuuden välisen tiedonsiirron. Viestintäprotokollan voi toteuttaa jokin laite, ohjelma tai yhdistelmä molempia. (Communications protocol 2017.)

TCP/IP

Transport Control Protocol/Internet Protocol. TCP/IP on internetin käyttämä joukko viestintäprotokollia. TCP/IP tarjoaa luotettavan tietoliikenteen eri päätepisteiden välille määrittelemällä, miten data tulisi paketoida, osoittaa, lähettää ja vastaanottaa verkossa. Toiminnallisuus on organisoitu neljään eri tasoon, joiden avulla kaikki joukkoon kuuluvat protokollat lajitellaan. (Internet Protocol Suite 2017.)

Kuvio 4 kuvaa TCP/IP-mallin tasot ja sen käyttämiä teknologioista.



Kuvio 4. TCP/IP-tasot ja osa niihin kuuluvista protokollista

TLS

Transport Layer Security ja sen edeltäjä SSL eli Secure Socket Layer ovat suosittuja salausprotokollia, jotka mahdollistavat tietoliikenteen salauksen (Transport Layer Security 2017).

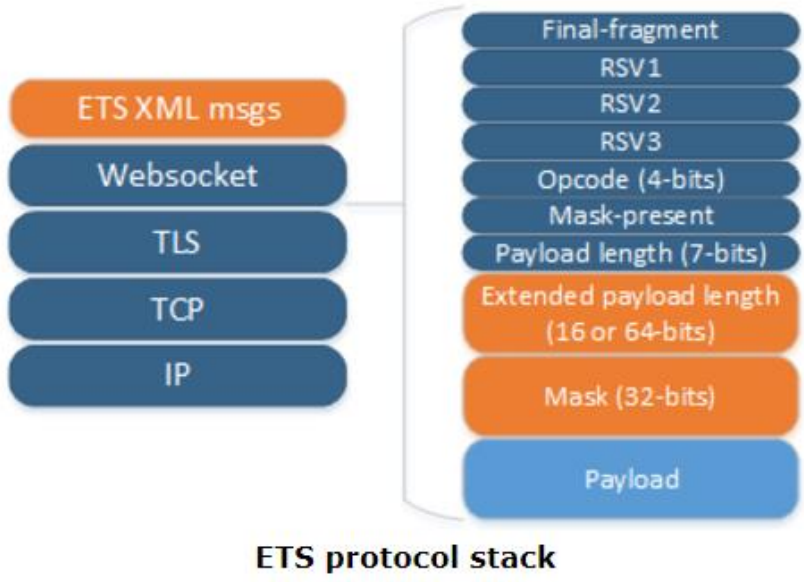
Websocket

Tarjoaa kaksisuuntaisen liikenteen mahdollistavia kanavia yhden TCP-yhteyden yli. Websocketin mahdollistaa asiakkaan ja palvelimen edestakaisen tiedon välityksen pitäen yhteyden auki. Data on minimaalisesti kehystetty pienellä ylätunnisteella jota seuraa viestin tietosisältö (engl. payload). (ETS Framework n.d.)

ETS (Event Transfer System)

Combitech Oy:n ohjelmistokehys, joka tarjoaa rajapinnat ja palvelut palvelujen löytämiselle, palvelujen konfiguroinnille, palvelujen suorittamiselle ja ylläpidolle sekä tämän projektin kannalta tärkeimmän eli Websocketiin ja XML-viesteihin pohjautuvan turvallisen kommunikaation. ETS-viestit toimitetaan verkon yli websocket payloadina. (ETS Framework n.d.)

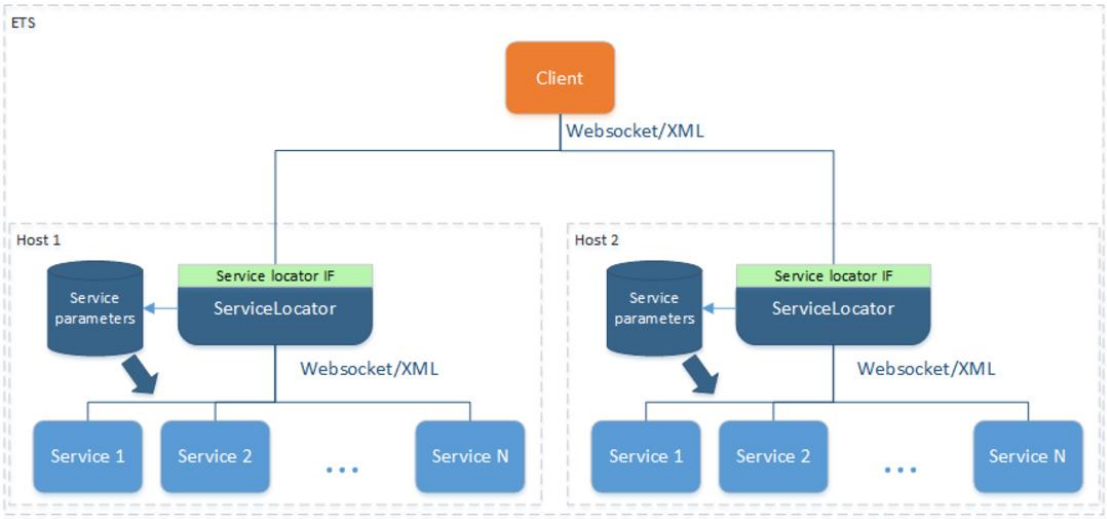
Kuviossa 5 on kuvattu miten ETS-järjestelmä rakentuu aiemmin käsiteltyjen teknologioiden päälle.



Kuvio 5. ETS-järjestelmän protokolla pino ja Websocket-taso (ETS Framework n.d.)

ETS käyttää asiakas-palvelin mallia. Palvelin kuuntelee palvelin-sockettia sille määritellyssä portissa, johon asiakkaat yhdistävät. Jokaisella yhteydellä voi olla useita roolimäärityksiä. Roolit ovat palvelimen työkalu asiakas-yhteyksien kategorisoimiseen. Palvelu määrittelee dokumentaatiossaan, minkälaisia rooleja se tukee. Asiakas voi tämän dokumentaation avulla päättää mitä rooleja se määrittää yhteyksilleen. (ETS Framework n.d.)

Kuviossa 6 on kuvattu miten sovellukset käyttävät ETS-järjestelmää yksinkertaistettuna. Opinnäytetyössä toteutettu sovellus toimii kuviossa esitettynä palveluna (engl. Service) ja siihen yhdistävätä audiolähteet ja vastaanottimet toimivat asiakkaina (engl. Client).



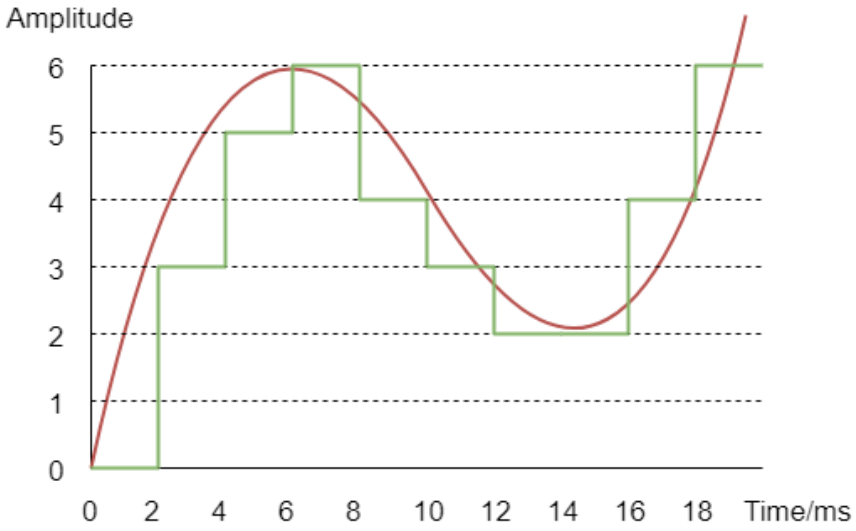
Kuvio 6. Yksinkertaistettu kuvaus ETS-järjestelmän toiminnasta (ETS Framework n.d.)

3.11 Audio

Audio tarkoittaa tuotettua ääntä. Audiota voi kuvata audiosignaaleina. Audiosignaalilla on ihmisen kuuloalueelle osuva taajuus. Perinteisesti audiosignaali esitetään sen analogisessa muodossa eli ääniaaltojen mukaisilla jännitteenmuutoksilla. (Audio signal 2017.)

Yhä useammin audiosignaali esitetään digitaalisessa muodossa. Yksi usein käytetty tapa esittää audiosignaalia digitaalisessa muodossa on pulssikoodimodulaatio tai PCM (engl. pulse-code modulation). PCM perustuu analogisesta signaalista tasaisin väliajoin otettaviin näytteisiin (engl. sample). Näytteen arvo näytteenottohetkellä vastaa reaalilukua. Tämä reaaliluku likimääräistetään vastaamaan sitä lähinnä olevaa digitaalisen portaan arvoa joka on kokonaisluku. PCM-striimin täsmäävyyden alkuperäiseen analogiseen signaaliin voi määrittää näytteenottotaajuuden ja yksittäisen näytteen bittisyvyyden (engl. bit depth) avulla. Bittisyvyys kertoo kuinka monta bittiä tietoa yhteen näytteeseen mahtuu. (Pulse-code modulation 2017.)

Kuviossa 7 on punaisella esitetty analoginen signaali. Vihreä kuvaa samaa signaalia muunnettuna digitaaliseen muotoon pulssikoodimodulaation avulla.



Kuvio 7. Analogisen signaalin muuntaminen digitaaliseksi PCM:n avulla

4 Toteutus

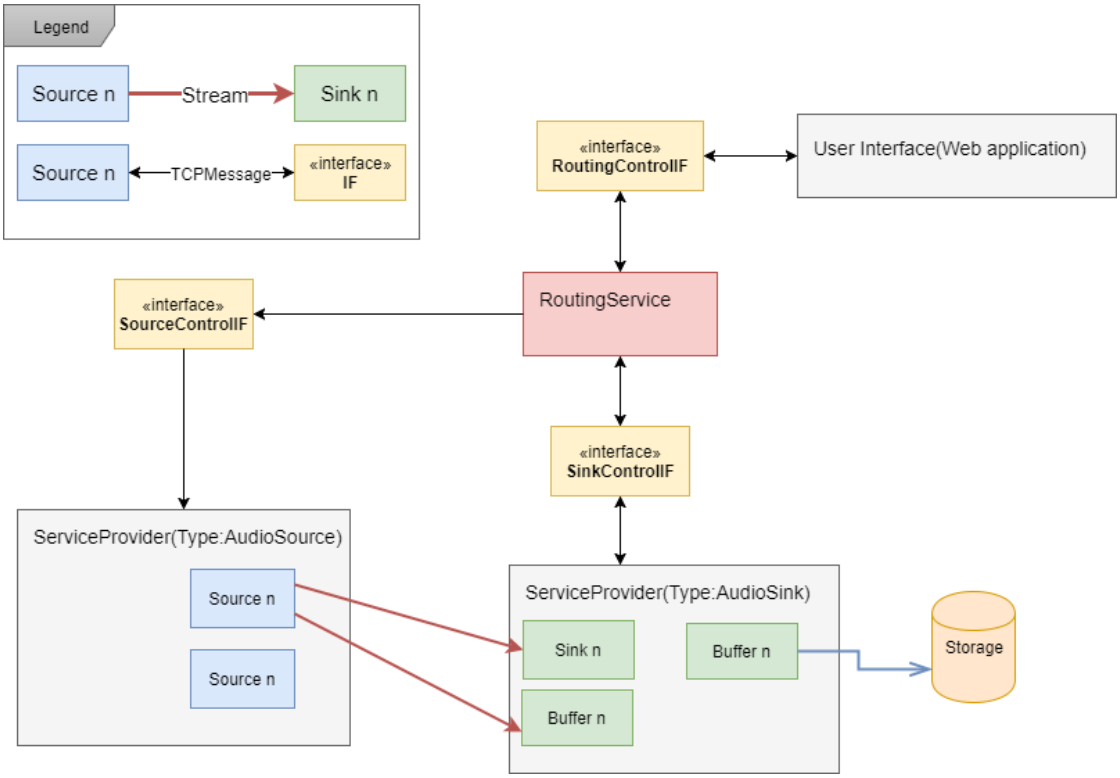
4.1 Yleistä

Opinnäytetyö toteutettiin osana pientä neljän henkilön ohjelmistoprojektia. Projektiryhmään kuului opinnäytetyön tekijän lisäksi toinen opiskelija, jonka työ keskittyy audion käsittelyyn vastaanottavassa päässä, toimeksiantajan Technical Lead Timo Mustonen, joka toimi projektipäällikön ja tuotteen omistajan kaltaisessa roolissa, ja Software Specialist Jani Honkonen, joka toteutti käyttöliittymän. Toteutuksen aikana teknisenä tukena oli koko Combitech Oy:n sisältä löytyvä kokemus ja asiantuntemus.

Projekti toteutettiin Scrum-mallin mukaisesti käyttäen kahden viikon mittaisia sprinttejä, jotka päättyivät aina tehdyn työn esittelyyn. Useimmiten näiden demojen yleisönä toimi toimeksiantajan muut työntekijät, jotka olivat erittäin kiinnostuneita työn edistymisestä ja tuloksista.

4.2 Ohjelmisto

Opinnäytetyössä toteutettu sovellus oli osa ohjelmistoa. Kuviossa 8 on ohjelmiston kokonaiskuva yksinkertaistettuna. Periaatteena on, että audiota lähettävät ja vastaanottavat sovellukset ovat ServiceProvidereita, joilla on yksi tai useampi audiolähde tai audiovastaanotin. Yksi audiovastaanotin (engl. sink) voi ottaa vastaan vain yhtä audiovirtaa kerrallaan, mutta yksi audiolähde (engl. source) pystyy lähettämään audiovirtaa usealla audiovastaanottimelle yhtä aikaa. RoutingService pystyy ohjailemaan audiolähteiden ja vastaanottimien toimintaa rajapintojen kautta. Käyttöliittymän avulla pystyy ohjailemaan RoutingServicen toimintaa.



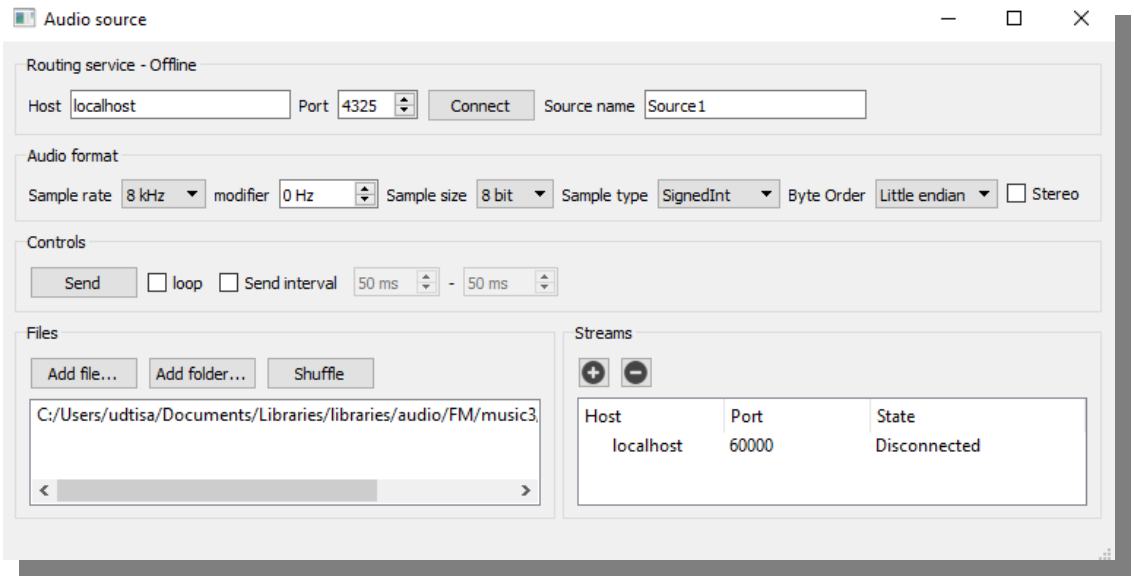
Kuvio 8. Ohjelmiston komponentit

4.3 Testisovellukset

4.3.1 AudioSource

AudioSource-sovellus on Timo Mustosen toteuttama sovellus, jonka tarkoitus on mahdollistaa RoutingService-sovelluksen testaus ja vianetsintä (engl. debuggaus). Sovelluksella luodaan Websocket-yhteys ensin RoutingService-sovellukseen ja sen antamien tietojen mukaan audiota vastaanottavaan sovellukseen, jolle se alkaa lähettämään audiovirtaa luomansa yhteyden ylitse. Sovellus voi lähettää saman datan useammalle lähteelle (esim. toistoon ja tallennukseen yhtä aikaa). Yhteyksiä audiovastaanottimille voidaan luoda ja poistaa myös manuaalisesti kuvion 9 mukaisen käyttö-

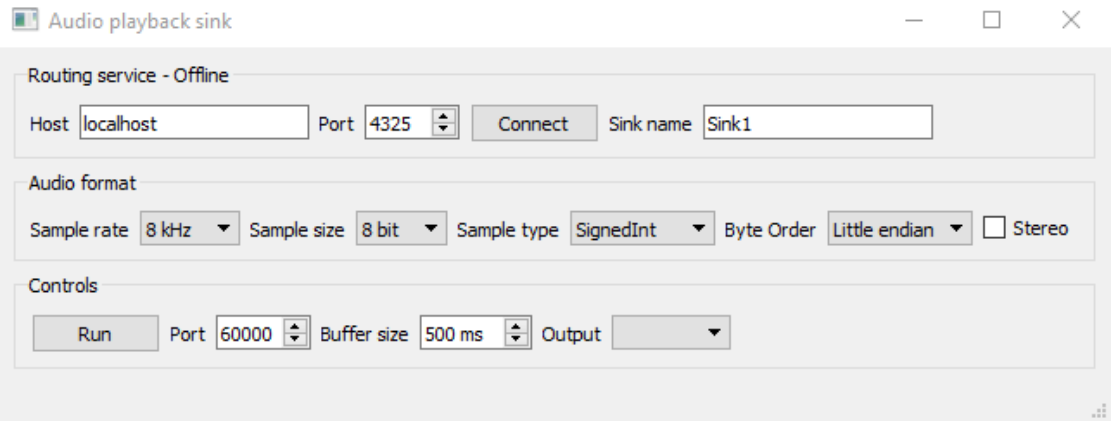
liittymän avulla. Sovelluksella voidaan määrittää lähtevän datan formaatti. Tämän formaatin kuuluu olla sama kuin vastaanottimella, jottei lähetetty data vääristy vastaanottavassa päässä.



Kuvio 9. AudioSource-sovelluksen käyttöliittymä

4.3.2 AudioPlaybackSink

AudioPlaybackSink on sovellus joka vastaanottaa audiovirtaa Websocket-yhteyden yli ja soittaa sitä järjestelmän audioulostuloista kuten tietokoneen kaiuttimista. Sovelluksella voidaan ottaa Websocket yhteys ohjauspalveluun, joka ohjaa sille audiovirran automaattisesti, mikäli sopivia AudioSource-sovelluksia on saatavilla. Mikäli AudioSourcelta ohjataan audiovirta manuaalisesti AudioPlaybackSinkille, on audioformaatin oltava molemmissa sama, muuten ääni vääristyy. AudioPlaybackSinkin audioformaattia voi manipuloida manuaalisesti kuvion 10 mukaisen käyttöliittymän avulla. Ohjauspalvelu tarkastaa, että audioformaatit täsmäävät, ennen kuin ohjaa audiovirran automaattisesti AudioSourcelta AudioPlaybackSinkille.



Kuvio 10. AudioPlaybackSink-sovelluksen käyttöliittymä

4.4 Ohjausrajapinnat

Ohjausrajapinnat toteutettiin siten, että ohjelmiston jokaiselle ohjattavalle sovellustyypille oli oma rajapinta. Nämä sovellustyypit olivat audiolähde, audiovastaanotin ja RoutingService. Käyttöliittymän toimintaa ei ollut tarkoitus ohjailla, joten käyttöliittymälle ei ollut omaa ohjausrajapintaa vaan se käyttää viestien lähettämiseen ja vastaanottamiseen RoutingControlIF-ohjausrajapintaa.

4.4.1 RoutingControlIF

RoutingControlIF tarjoaa RoutingServicen ohjausrajapinnan. Käyttöliittymä käyttää rajapintaa ohjaamaan RoutingServicen toimintaa ja luomaan tai poistamaan audiovirtoja. Myös Audiolähteet ja vastaanottimet käyttävät tätä rajapintaa rekisteröityessään RoutingServicelle. Käytännössä RoutingServicen suuntaan tapahtuvaan kommunikointiin käytetään tätä rajapintaa ja RoutingService käyttää rajapintaa kertoakseen käyttöliittymälle eri tapahtumista.

4.4.2 SourceControlIF

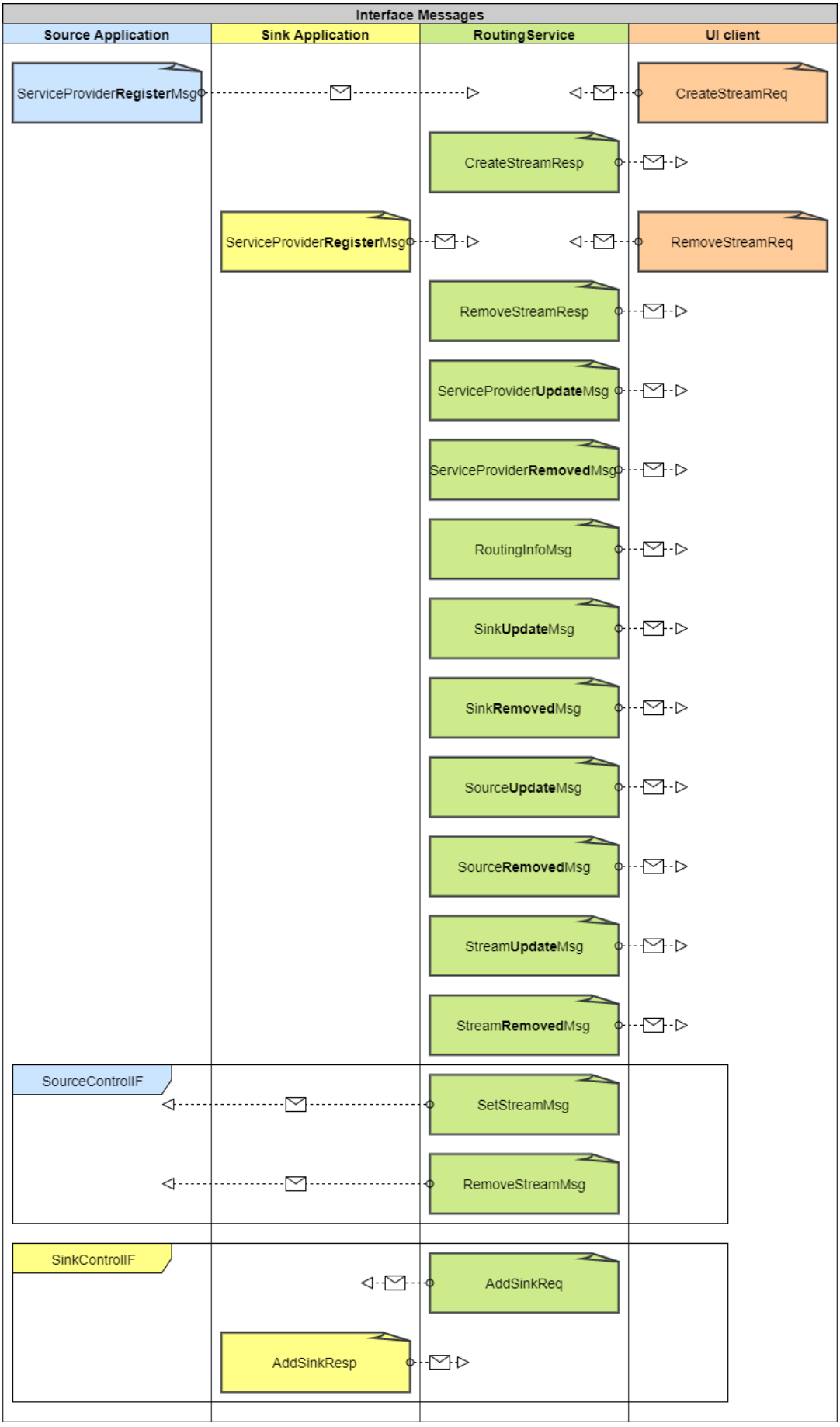
SourceControlIF tarjoaa ohjausrajapinnan RoutingServicen ja audiolähteiden välille. RoutingService ohjaa tämän rajapinnan avulla audiolähde-tyyppisiä sovelluksia luomaan tai poistamaan audiovirtoja.

4.4.3 SinkControlIF

SinkControlIF tarjoaa ohjausrajapinnan RoutingServicen ja audiovastaanottimien välille. Tämän rajapinnan avulla RoutingService voi pyytää audiovastaanotin-tyyppisiä sovelluksia luomaan uusia vastaanottimia audiolähteille, joille ei ole sopivaa vastaanotinta.

4.4.4 Viestit

Ohjausrajapinnat toteutettiin viestien avulla. Ohjattavalle sovellukselle lähetetään sen ohjausrajapinnassa määritelty viesti, jonka avulla se ohjataan suorittamaan haluttu toiminto. RoutingService käyttää RoutingControlIF rajapintaa myös ilmoittamaan käyttöliitymille sen tietopohjassa tapahtuvat muutokset. Kuviossa 11 on kuvattu sovellusten väliset viestit ja niiden lähetyssuunnat. Kuvion 11 alalaidan kehykset kuvaavat audolähde- ja audiovastaanotin-tyyppisten sovellusten ohjausrajapinnoista löytyviä viestejä. Kaikki näiden kehysten ulkopuolella olevat viestit kuuluvat RoutingServicen ohjausrajapintaan.



Kuvio 11. Ohjausrajapintojen sisältämät viestit ja niiden lähetyssuunnat

4.4.5 ETS-järjestelmän käyttö rajapintojen toteutuksessa

Ohjausrajapintojen toteutuksessa käytettiin hyväksi ETS-järjestelmästä löytyvää tapahtumien (engl. event) käsittelyyn tarkoitettua toimintoa. Kaikki viestit ovat ennalta määriteltyjä luokkia. ETS-järjestelmä muuttaa luokan olion attribuutteineen XML-formaattiin, josta se muutetaan vastaanottavassa päässä taas luokan olioksi.

Kuviossa 12 on kuvattu esimerkki ohjausrajapintaan määritetystä viestiluokasta. Ohjelmiston käyttämät viestiluokat ovat rakenteeltaan samankaltaisia.

```
class SetStreamMsg : public RoutingBase::MsgBase
{
public:
    RoutingIFData::AudioSink sink;  /*!< Target sink */
    string sourceId;    /*!< Id of source(needed if single provider has many sources active) */
    string targetHost; /*!< Address of sink for creating TCP/IP connection */
    string creator; /*!< Indicates who made the stream(Rule name, or name of client) */

    /*!< Method used by ETS to convert onject of this class to XML */
    void Stream(AbstractStreamer& p){
        MsgBase::Stream(p); /*!< msgTime inherited from MsgBase */
        p("sink", sink);
        p("sourceId", sourceId);
        p("targetHost", targetHost);
        p("creator", creator);
    }
};
```

Kuvio 12. SetStreamMsg-luokka

Kuvioissa 13 – 18 on esimerkki viestirajapinnan käytöstä ETS:n ja Qt:sta löytyvä Signals & Slots-menetelmän avulla. Kuviossa 13 luodaan SetStreamMsg luokasta olio setMsg. Tämä olio alustetaan parametrina saaduilla tiedoilla ja järjestelmän sen hetkisellä kellonajalla. Alustuksen jälkeen kutsutaan routingData-luokan funktiota addStream, joka lisää luodun audiovirran tietopohjaan ja palauttaa sille asetetun tunnisteen arvon. Funktio sendStreamUpdateMessage() ilmoittaa kaikille käyttöliittymä-asiakkaille (engl. UI-client) tapahtuneesta muutoksesta. Lopulta kutsutaan ETS:n funktiota SendEvent(connID, setMsg), jossa parametri connID on vastaanottavan AudioSource-sovelluksen yksilöllinen tunnus ja parametri setMsg on aiemmin alustettu olio SetStreamMsg-luokasta. Tämän SetStreamMsg-viestin avulla RoutingService-sovellus ohjaa audiolähteen lähettämään audiovirtaa viestissä määriteltyyn audiovastaanottimeen.

```
void RoutingService::sendSetStreamMessage(string connID, string sourceId,
        RoutingIFData::AudioSink sink, string host, string creator){

    if (d->ruleEngine.formatMatch(source.value().format, sink.format)){
        SourceControlIF::SetStreamMsg setMsg;
        setMsg.sink = sink;
        setMsg.sourceId = sourceId;
        setMsg.targetHost = host;
        setMsg.creator = creator;
        setMsg.msgTime = QDateTime::currentMSecsSinceEpoch();
        string streamId = d->routingData.addStream(sourceId, sink.uuid, creator);
        sendStreamUpdateMessage(streamId);
        SendEvent(connID, setMsg);
    }else {
        qDebug() << "Formats did not match";
    }
}
```

Kuvio 13. Esimerkki SetStreamMsg:n initialisoinnista, sen lähettämisestä

Kuviossa 14 nähdään aiemmin alustettu setMsg-olio tietoineen XML-formaatissa.

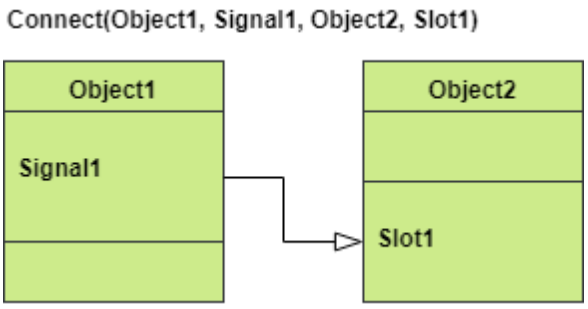
```
<SourceControlIF::SetStreamMsg
    xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">
    <msgTime>1500630156185</msgTime>
    <sink>
        <uuid>6e03ebf1-517d-419a-97cf-e5f7b4d13c40</uuid>
        <serviceUuid>1bdf8b75-9b39-44d8-9bb5-65a2556fda43</serviceUuid>
        <name>Sink1</name>
        <type>playback</type>
        <protocol />
        <port>60000</port>
        <format>
            <codec>audio/pcm</codec>
            <sampleRate>8000</sampleRate>
            <sampleSize>8</sampleSize>
            <sampleType>1</sampleType>
            <channelCount>1</channelCount>
            <byteOrder>1</byteOrder>
        </format>
    </sink>
    <sourceId>b31c8198-76ad-4ff3-83f6-ed34f21701be</sourceId>
    <targetHost>WLG00055</targetHost>
    <creator>Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
        (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36</creator>
</SourceControlIF::SetStreamMsg>
```

Kuvio 14. SetStreamMsg:n sisältö XML-formaatissa

Qt:n Signals & Slots

Yksi Qt:n keskeisimmistä toiminnoista on Signals & Slots. Toiminnon avulla yhdistetään sovelluksen eri olioiden toiminnallisuus toisiinsa. Esimerkiksi kun käyttöliittymästä klikataan **Close**-nappia, halutaan myös kutsua **close()**-funktiota. (Signals & Slots n.d.)

Kuvio 15 kuvaa yksinkertaistettuna signaalin ja slotin yhdistämisen. Signaali lähetetään, kun tietty tapahtuma toteutuu. Tähän signaaliin yhdistetty funktio kutsutaan aina vastauksena lähetettyyn signaaliin.



Kuvio 15. Malli kahden olion yhdistämisestä Signals & Slots-toiminnolla

Kuviot 15-17 sisältävät esimerkin Signals & Slots-toiminnon käytöstä projektin toteutuksessa. Kuvio 16 kuvaa AudioSource-sovelluksen RoutingServiceClient-luokan funktion, joka vastaanottaa ETS-tapahtuman SetStreamMsg. Aina kun RoutingServiceClient-olio vastaanottaa SetStreamMsg-tapahtuman, se lähettää (engl. emit) signaalin, joka sisältää ETS-tapahtuman mukana tulleen viestin tiedot.

```
20  void RoutingServiceClient::ProcessETSEvent(shared_ptr<SourceControlIF::SetStreamMsg> e,
21                                              const string connID){
22      emit messageReceived(e->msgTime, e->sourceId, e->sink, e->targetHost, e->creator);
23  }
```

Kuvio 16. ETS tapahtuman vastaanottaminen ja messageReceived signaalin lähettäminen

Kuvio 17 kuvaa kahden olion toimintojen yhdistämisen käytännössä. connect()-funktio yhdistää rsClient-olion messageReceived-signaalin, this(AudioSourceMainWindow)-olion on_messageReceived-funktioon. Aina kun rsClient lähettää messageReceived-signaalin, AudioSourceMainWindow kutsuu on_messageReceived-funktiota.

```
168     d->rsClient.reset(new RoutingServiceClient);
169     connect(d->rsClient.get(), &RoutingServiceClient::messageReceived,
170             this, &AudioSourceMainWindow::on_messageReceived);
171
```

Kuvio 17. Signalin ja Slotin yhdistäminen

Kuvio 18 kuvaa on_messageReceived-funktion, joka käsittelee vastaanotetun viestin. Kuviosta voidaan huomata, että sen parametrit täytyy esittää samassa järjestyksessä kuin kuvion 16 kuvaamassa signaalin lähetyksessä.

```
235 void AudioSourceMainWindow::on_messageReceived(int msgTime,
236                                                 string sourceId,
237                                                 RoutingIFData::AudioSink sink,
238                                                 string targetHost,
239                                                 string creator){
240     qDebug() << msgTime << endl << "Received SetStreamMSG";
241
242     string uuid = DlStringUUIDCreate();
243     QString streamID = QString::fromStdString(uuid);
244     QString host = QString::fromStdString(targetHost);
245
246     StreamCollection::StreamData sData;
247     sData.insert(StreamCollection::Host, host);
248     sData.insert(StreamCollection::Port, sink.port);
249     sData.insert(StreamCollection::State, tr("Disconnected"));
250
251     qStreams->insert(streamID, sData);
252
253     shared_ptr<StreamContext> stream(new StreamContext);
254     stream->id = streamID;
255     stream->host = host;
256     stream->sink = sink;
257     stream->creator = QString::fromStdString(creator);
258     connect(&stream->socket, &QTcpSocket::stateChanged, this,
259             &AudioSourceMainWindow::on_socket_stateChanged);
260     stream->socket.connectToHost(stream->host, stream->sink.port);
261     //stream->socket.waitForConnected(5000);
262     d->streams.insert(streamID, stream);
263 }
```

Kuvio 18. on_messageReceived-funktio käsittelee vastaanotetun viestin

Signals & Slots on siis erittäin näppärä ja suhteellisen helppokäyttöinen toiminto ja ainakin toimeksiantajan kokeneempien ohjelmoijien erittäin hyvänä pitämä ratkaisu.

4.5 RoutingService

4.5.1 Yleistä

RoutingService koostuu kolmesta komponentista. Komponentit ovat ohjauspalvelu (RoutingService), tietopohja (RoutingData) ja päättelykone (RoutingRuleEngine). Yleensä olio-ohjelmoinnissa pyritään luomaan täysin toisistaan riippumattomia komponentteja. RoutingServiceä voi käyttää käyttöliittymän avulla manuaaliseen ohjaukseen myös ilman RoutingRuleEngineä. RoutingService säilöö RoutingData-luokan olioon tiedot yhdistäneistä sovelluksista ja niiden välisistä audiovirroista, joten se ei pysty toimimaan halutulla tavalla ilman RoutingData-luokkaa.

Kuvio 21 kuvaa RoutingService-sovelluksen luokkakaavion. RoutingServicen toiminta perustuu pitkälti liitteessä 2 kuvattuihin abstrakteihin tietotyyppeihin. Abstrakti tietotyyppi tarkoittaa ohjelmoijan määrittelemää tietotyyppiä, joka voi sisältää muita abstrakteja tai primitiivisiä (integer, string jne.) tietotyyppejä. Abstraktit tietotyypit ovat yksi olio-ohjelmointimallin vahvuuksista.

Luokkakaaviota tarkastelemalla selviää eri luokkien vastuualueet ja niiden väliset riippuvuussuhteet. RoutingService-luokka on riippuvainen RoutingData-luokasta ja sille määritetystä kuvion 19 mukaisesta Private-luokasta, jota käytetään hyväksi RoutingData ja RoutingRuleEngine luokkien funktioita ja muuttujia käytettäessä kuviossa 20 esitetyllä tavalla.

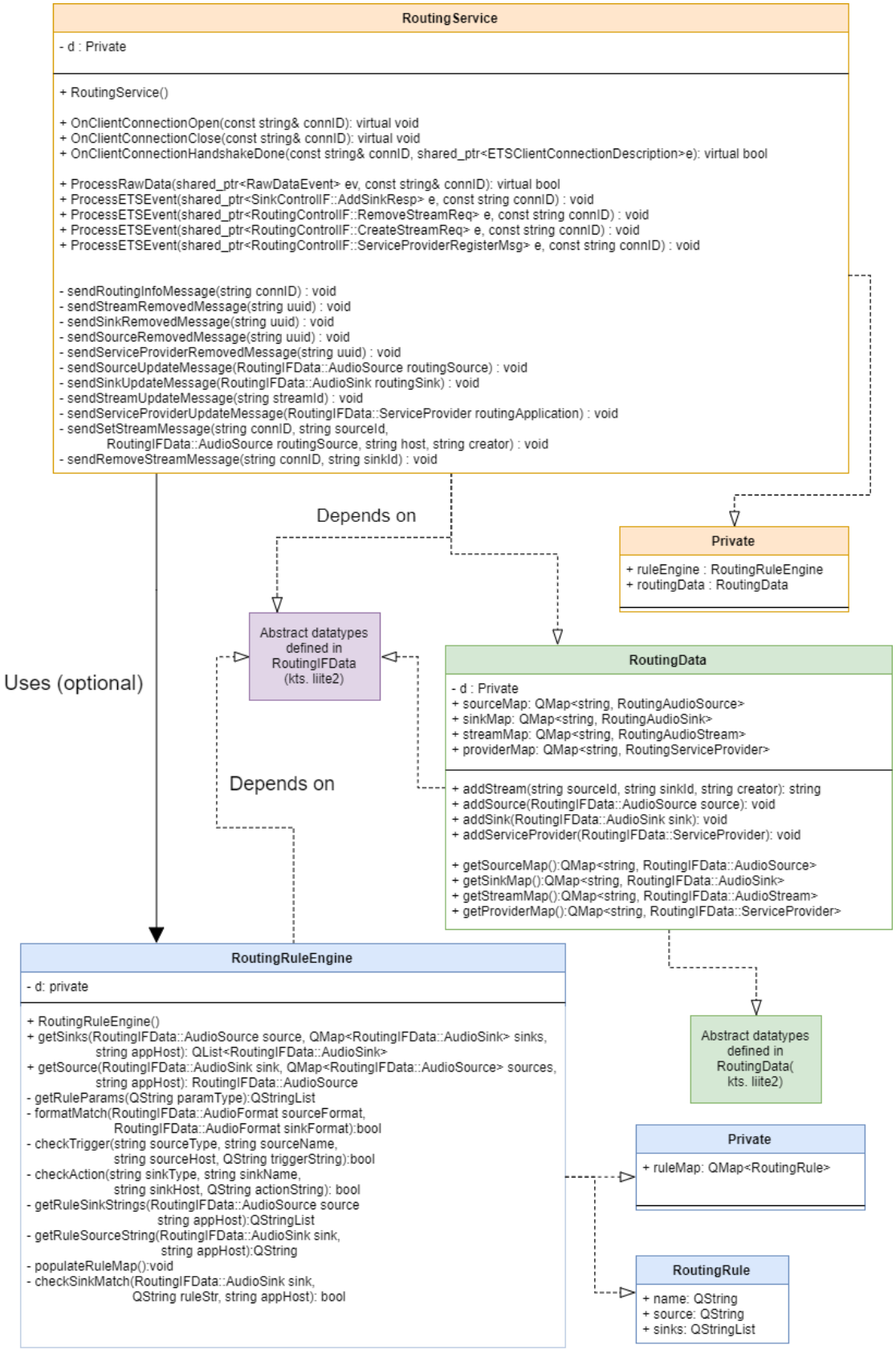
```
79  private:
80      class RoutingService::Private{
81      public:
82          RoutingRuleEngine ruleEngine;
83          RoutingData routingData;
84
85      };
86      std::shared_ptr<Private> d;
87  };
```

Kuvio 19. Private-luokan määrittely

```
339 RoutingService::RoutingService(){
340     d.reset(new Private);
341 }
342
343 void RoutingService::ProcessETSEvent(shared_ptr<SinkControlIF::AddSinkResp> e,
344     if (e->status == true){
345         d->routingData.addSink(e->sink);
346         for (auto s : d->routingData.sourceMap){
347             if (s.uuid == e->sourceUuid){
348                 auto i = d->routingData.providerMap.find(s.serviceUuid);
349                 if (i != d->routingData.providerMap.end()){
350                     sendSetStreamMessage(i.value().connID, s.uuid, e->sink, e-
351                 }else {
352                     qDebug() << "Service not found";
353                 }
354             }
355         }
356     }
357 }
358
```

Kuvio 20. Esimerkki Private-luokan käytöstä

Funktioiden näkyvyyksiä tarkastelemalla nähdään, että suurin osa RoutingService-luokan funktioista on tarkoitettu käytettäväksi vain luokan sisällä. Sama pätee RoutingRuleEngine-luokkaan, jolla on konstruktorin lisäksi vain kaksi julkista funktiota, jotka on tarkoitettu RoutingService-luokan kutsuttavaksi audiolähteen tai audiovastaanottimen yhdistäessä palveluun. Tällaisella ajattelulla on pyritty selkiyttämään luokkien vastuualueita ja toiminnallisuutta.



Kuvio 21. Kehitystyön tukenä käytetty RoutingService-sovelluksen luokkakaavio

RoutingServicen tarkoitus on toimia palveluna, johon audiolähteet ja audiovastaanottimet voivat yhdistää niiden välisten audiovirtojen ohjauksen mahdollistamiseksi. RoutingServiceä voi käyttää audiovirtojen ohjaamiseen kahdella tavalla: käyttöliittymän kautta manuaalisesti tai sääntöjen avulla automaattisesti. Molemmissa tapauksissa RoutingService käyttää SourceControlIF-ohjausrajapintaa kertoakseen mihin audiovastaanottimiin audiolähteen tulisi lähettää tuottamansa audiovirta.

RoutingService käyttää hyväkseen päättelykonetta luodakseen yhteyksiä audiolähteiden ja audiovastaanottimien välille. RoutingService viestii reaaliajassa tilastaan ohjauspalveluun yhdistäneille käyttöliittymille.

RoutingService toimii nimensä mukaisesti palveluna muille sovelluksille. ETS sisältää monia palveluna toimimiseen vaadittavia ominaisuuksia. RoutingService periytyy ETS:stä löytyvästä ETSServiceBaseQt-luokasta. Luokka tarjoaa RoutingServicen käyttöön Websocket-yhteyksien avaamisen asiakkaiden ja RoutingServicen välille sekä ETS-tapahtumien lähettämiseen ja vastaanottamiseen tarvittavan toiminnallisuuden.

RoutingServicen käynnistyessä sovellus jää kuuntelemaan sille määrättyyn porttiin saapuvia yhteyksiä. Asiakkaan muodostaessa yhteyden RoutingServiceen se lähettää viestin, jonka avulla RoutingService voi, arvioida hyväksyykö yhteyden vai ei. RoutingService hyväksyy yhteydet vain RoutingControlIF rajapintaa käyttäviltä sovelluksilta. RoutingControlIF rajapinnassa on määritelty omat roolit jokaiselle hyväksyttävälle sovellustyypille eli audiolähteelle, audiovastaanottimelle ja käyttöliittymälle. Tämän roolin avulla RoutingService myös reagoi oikealla tavalla saapuvaan yhteyteen. Esimerkiksi käyttöliittymä-asiakkaan yhdistäessä yhteys hyväksytään ja asiakkeella vastataan nykyiset audiolähteet, audiovastaanottimet ja niiden väliset audiovirrat sisältävällä viestillä.

4.5.2 Program Flow

Kuviossa 22 on kuvattu RoutingService-sovelluksen toiminta. RoutingService reagoi eri tavoin riippuen rekisteröityneen sovelluksen tyypistä. RoutingService-sovellus käynnistetään komentoriviltä, jolloin se jää odottamaan viestejä erilaisilta rajapintoja käyttäviltä asiakas-sovelluksilta. Jokainen sovellus rekisteröityy RoutingServicelle

viestillä, joka sisältää audiovirtojen ohjaamiseen tarvittavia tietoja rekisteröityneestä sovelluksesta, kuten tyyppi ja tuetut formaatit.

Audiolähde (Source)

Audiolähteen rekisteröityessä RoutingService kutsuu päättelykonetta, joka palauttaa johonkin sääntöön täsmäävän tietopohjassa olevan audiovastaanottimen tiedot. Tämän jälkeen RoutingService luo audiovirrat audiolähteen ja audiovastaanottimien välille. Mikäli täsmääviä vastaanottimia ei löydy, lähteen tiedot lisätään tietopohjaan, ja se jää odottamaan ohjausta.

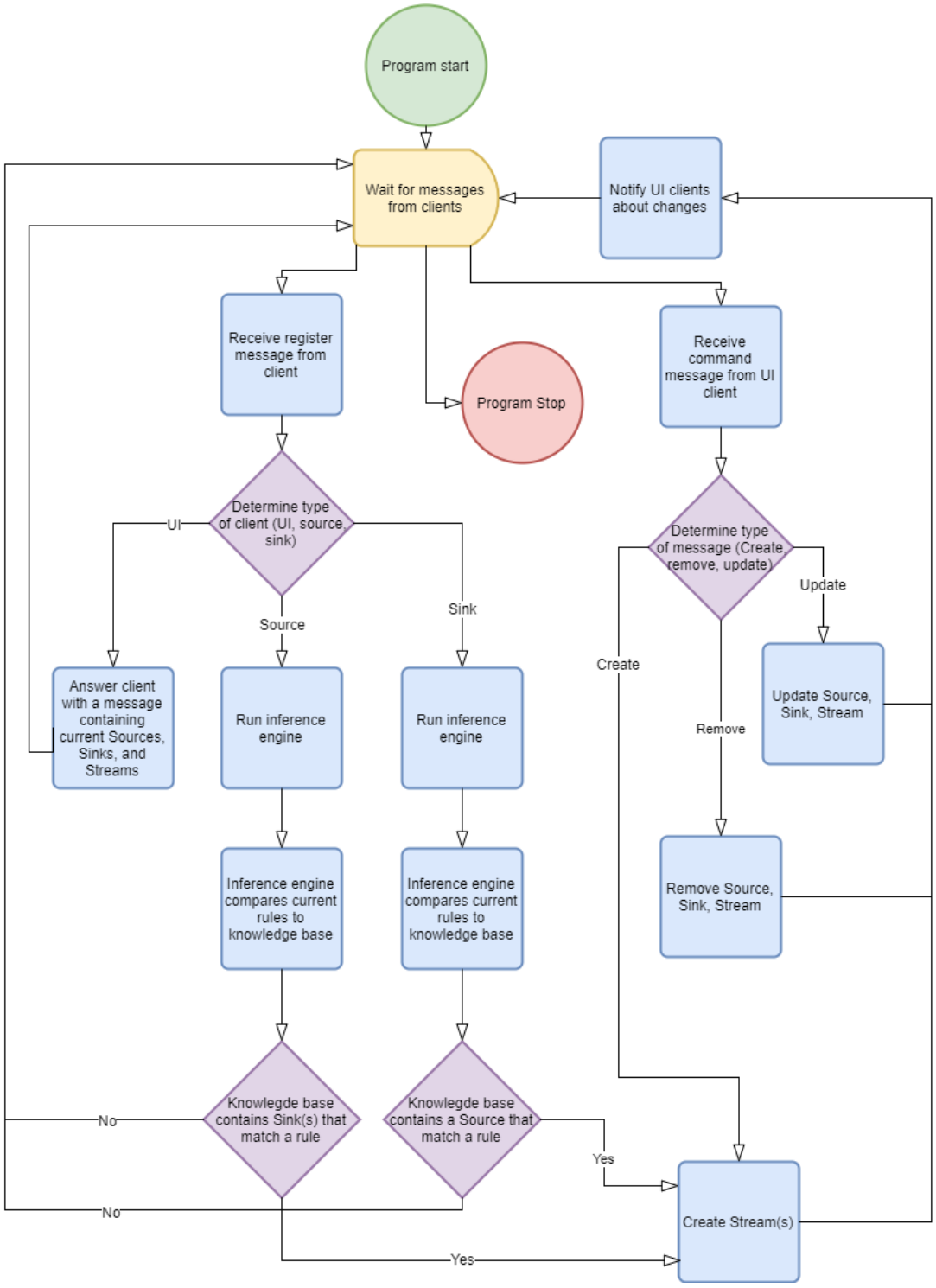
Audiovastaanotin (Sink)

Audiovastaanottimen rekisteröityessä RoutingService kutsuu päättelykonetta, joka palauttaa johonkin sääntöön täsmäävän tietopohjassa olevan audiolähteen tiedot, minkä jälkeen RoutingService ohjaa audiolähteen lähettämään audiovirtaa rekisteröityneelle audiovastaanottimelle. Mikäli täsmääviä lähteitä ei löydy, vastaanottimen tiedot lisätään tietopohjaan, ja se jää odottamaan audiovirtaa.

Käyttöliittymä (UI)

Käyttöliittymä-asiakkaan rekisteröityessä sen tiedot lisätään tietopohjaan ja sille vastataan viestillä, joka sisältää kaikki tietopohjan sisältämät audiolähteet, audiovastaanottimet ja audiovirrat. Tämän viestin avulla käyttöliittymä voi visualisoida reaaliaikaisen tilanteen heti rekisteröidyttyään. RoutingService kertoo käyttöliittymä-asiakkaille jokaisen tietopohjassa tapahtuvan muutoksen.

Käyttäjä voi ohjata käyttöliittymän avulla RoutingServicen toimintaa. Käyttöliittymän kautta voi luoda audiovirran audiolähteen ja vastaanottimen välille, poistaa audiovirtoja tai päivittää audiovirran nykyisen päämäärän joksikin toiseksi. Tämän lisäksi rekisteröityneet sovellukset voidaan poistaa tietopohjasta käyttöliittymän avulla, mutta kuviosta 22 poiketen sovelluksien tietojen päivittämiseen liittyvää toimintoa ei toteutettu.



Kuvio 22. RoutingService-sovelluksen program flow-kaavio

4.6 RoutingRuleEngine

RoutingRuleEngine on nimensä (engl. rule engine, suom. sääntökone) vastaisesti itse asiassa päättelykone, joka pystyy toimimaan molemmissa sekä eteenpäin että taaksepäin ketjuttavassa tilassa. RoutingRuleEngine-luokan tarkoituksena on parsia säännöt XML-tiedostosta käsiteltävään muotoon. Uuden lähteen tai vastaanottimien liittyessä RoutingRuleEngine-luokan avulla tarkistetaan, täsmääkö juuri liittynyt sovellus yhteenkään sääntöön.

RoutingRuleEngine luokka suorittaa automaattisen ohjauksen vaatiman päättelyn. Tämä komponentti toimii järjestelmän tekoälynä. RoutingRuleEngine luokan toteutus oli projektin vaikein osuus, sillä vaikka valmiita toteutuksia on olemassa, niiden lähdekoodi on harvoin avointa.

Saatavilla olevan vähäisen tiedon takia RoutingRuleEngine luokkaa lähdettiin suunnittelemaan vaatimusmäärittelyä apuna käyttäen alusta alkaen. Vaatimusmäärittelyn mukaan säännöt tulee säilöä tiedostoon, joka on helposti sekä ihmisen että tietokoneen luettavissa. Sääntöjen parametrejä tulee pystyä yhdistelemään käyttäen tunnettuja operaattoreita AND (looginen JA), OR (looginen TAI) ja NOT (looginen EI). Tämän lisäksi tulisi olla mahdollisuus käyttää jokerimerkkejä (engl. wildcard) kuten * (mikä tahansa merkkijono) ja ? (mikä tahansa yksittäinen merkki).

Sääntöjen syntaksiin oli suunnitteluvaiheessa käytännössä kaksi vaihtoehtoa. Kuviot 23 ja 24 sisältävät yksinkertaisen säännön. Molemmissa kuvioissa säännön toiminto on sama: mikäli liittyvän AudioSource-sovelluksen hostname on jompikumpi OR-operaattoria ympäröivistä ehdoista, ohjataan datavirta Goal solmun ilmoittamassa osoitteessa sijaitsevaan AudioPlaybackSink-sovellukseen. Kuviossa 23 esitetään operaattori OR XML-solmuna (engl. XML-node), kun taas kuviossa 24 OR-operaattori on osana ehdon määrittävää merkkijonoa.

```
<Rule>
    <Name>Rule1</Name>
    <Condition>
        <Parameter>sourceHost=WLG00055<Parameter>
        <OR>
        <Parameter>sourceHost=WLG00048</Parameter>
    </Condition>
    <Goal>sinkHost=WLG00055</Goal>
</Rule>
```

Kuvio 23. Sääntösyntaksin ensimmäinen vaihtoehto

```
<Rule>
    <Name>Rule1</Name>
    <Condition>sourceHost=WLG00055 OR sourceHost=WLG00048</Condition>
    <Goal>sinkHost=WLG00055</Goal>
</Rule>
```

Kuvio 24. Sääntösyntaksin toinen vaihtoehto

Valintaan vaikutti käytännössä kaksi tekijää: luettavuus ja ohjelmallisen käsittelyn vaikeus. Lopullisen sovelluksen käyttäjät ovat insinöörejä, joten voidaan olettaa, että tavalliset operaattorit ovat heille tuttuja tai että käyttäjä oppii niiden käytön perehtymällä sovelluksen dokumentaatioon.

Vaihtoehto yksi saattaa äkkiseltään tuntua loogisemmalta vaihtoehdolta, mutta se sisältää tiettyjä heikkouksia verrattuna vaihtoehtoon kaksi. Vaihtoehto kaksi on opinnäytetyöntekijän mielestä helpompilukuinen etenkin monimutkaisempien sääntöjen suhteen. Vaikka projektin aikana ei toteutettu sääntöjen määritystä käyttöliittymän kautta, se oli mukana suunnittelussa. Vaihtoehdossa kaksi käyttäjältä voi kysyä yhden ehdon yhdellä tekstikentällä ja asettaa käyttäjän antama syöte suoraan XML-solmun tiedoksi, kun taas vaihtoehdossa yksi käyttäjän antamaa syötettä täytyisi parsia jollain tavalla.

Vaihtoehdon yksi käyttö hankaloittaisi myös sääntökoneen suorittamaa käsittelyä. Kuviossa 25 on esimerkki XML-tiedoston käsittelystä QDomNode-luokan avulla solmu solmulta. Kuviosta 25 nähdään, että jokainen eri nimen omaava solmu pakottaa lisäämään ehtoja ja silmukoita. Osa sääntökoneen päättelylogiikasta tulisi olla XML-tiedoston käsittelyvaiheessa. Opinnäytetyön tekijä halusi jakaa toiminnallisuuden siten, että RoutingRuleEnginestä löytyy funktio, joka lukee XML-tiedoston sovelluksen tietoon, ja funktio, joka vertaa tietopohjan sisältöä aiemmin luettuihin sääntöihin ja päättelee suoritettavan toiminnon.

```
29  void RoutingRuleEngine::populateRuleMap(){
30      QDir directory;
31      QString path = directory.homePath();
32      path += "/Documents/Projects/audio-routing-and-recording/Service/Src/Ro
33      QDomDocument doc("mydoc");
34      QFile file(path);
35
36      if (!file.open(QIODevice::ReadOnly))
37          return;
38      if (!doc.setContent(&file)) {
39          if (!doc.setContent(&file)) {
40              file.close();
41              return;
42          }
43          file.close();
44      }
45      QDomElement topElement = doc.documentElement();
46      QDomNode node = topElement.firstChild();// Rule
47      while (!node.isNull()){
48          QDomElement element = node.toElement();
49          if (element.tagName() == "Rule"){
50              RoutingRule rule;
51              rule.name = node.childNodes().at(0).firstChild().nodeValue();
52              if(!element.isNull()){
53                  QDomNode node1 = element.firstChild();
54                  while(!node1.isNull()){
55                      QDomElement element1 = node1.toElement();
56                      if (element1.tagName() == "Source"){
57                          rule.source = element1.text();
58                      }else if (element1.tagName() == "Sink") {
59                          rule.sinks.push_back(element1.text());
60                      }
61                      node1 = node1.nextSibling();
62                  }
63              }
64              d->ruleMap.insert(rule.name, rule);
65          }
66          node = node.nextSibling();
67      }
68  }
```

Kuvio 25. XML-tiedoston käsittelyä QDomNode-luokan avulla

RoutingRuleEnginen ensimmäisessä toimivassa prototyypissä suoritettavat toiminnot pääteltiin vain eteenpäin ketjuttamalla. Käytännössä audiovastaanotin-tyyppisen sovelluksen liittyessä se vain lisättiin osaksi tietopohjaa ja yhteys pidettiin auki. audiolähde-tyyppisen sovelluksen liittyessä sen tietoja verrattiin sääntöihin ja tietopohjassa oleviin audiovastaanottimiin. Säännön täsmätessä tietopohjan sisältöön ohjataan audiovirta säännön määrittelemiin audiovastaanottimiin. Silloin säännöt näyttivät Kuvion 26 mukaisilta. Mikäli liittyvä audiolähde täsmäsi Trigger-solmun sisältämiin parametreihin, tarkistettiin, löytyykö tietopohjasta Action-solmun sisältämiin parametreihin täsmäävää vastaanotinta. Molempien ehtojen täyttyessä ohjattiin audiovirta audiolähteeltä vastaanottimeen.

```
<Rule>
    <Name>Rule1</Name>
    <Trigger>sourceType=AudioSource AND sourceName=Source1</Trigger>
    <Action>sinkType=AudioPlaybackSink</Action>
</Rule>
```

Kuvio 26. Sääntö RoutingRuleEnginen alkutaipaleelta

RoutingRuleEnginen toimiessa hyvin eteenpäin ketjutettaessa sprintin suunnittelupalaverissa todettiin aiheelliseksi sääntökoneen käyttö myös audovastaanotin-tyyppisen sovelluksen liittyessä palveluun. Tämä muuttaa RoutingRuleEnginen kattamaan myös taaksepäin ketjutuksen ja tekee sääntökoneesta tarkemmin päättelykoneen sääntökone termin kattaessa vain eteenpäin ketjuttavat toteutukset. Sääntöjen solmujen termit eivät myöskään kuulostaneet enää tarkoituksenmukaiselta Actionin toteuttaessa Triggerin. Selkeyden vuoksi sääntöjen syntaksi muutettiin kuvion 27 mukaiseksi. Kuviossa 27 kuvattu sääntö toimii seuraavalla tavalla: Source-solmussa määriteltyihin parametreihin täsmäävän audiolähteen liittyessä RoutingServiceen, RoutingRuleEngine tarkistaa tietopohjan sisällön ja löytäessään täsmäävän vastaanottimen tai vastaanottimet ohjataan audiolähde lähettämään audiovirtaa niihin. Mikäli täsmääviä audiolähteitä ei löydy tietopohjasta, audiolähde rekisteröidään tietopohjaan ja yhteys pidetään auki. Kun jommassakummassa Sink-solmussa määriteltyihin parametreihin täsmäävä audiovastaanotin liittyy RoutingServiceen, RoutingRuleEngine tarkistaa tietopohjan ja

löytäessään täsmäävän audiolähteen ohjaa RoutingService tämän audiolähteen lähettämään audiovirtaa juuri liittyneelle audiovastaanottimelle.

```
43  <Rule>
44      <Name>Rule1</Name>
45      <Source>sourceType=buffer AND sourceHost=WLG00055</Source>
46      <Sink>sinkType=playback AND sinkHost=WLG00055</Sink>
47      <Sink>sinkType=playback AND sinkHost=WLG00044</Sink>
48  </Rule>
```

Kuvio 27. Sääntö viimeisimmässä muodossa

Säännön viimeisimmässä muodossa yksi Rule-solmu voi sisältää vain yhden Source-solmun, mutta useita Sink-solmuja. Tämä johtuu sovellustyyppien rakenteesta, sillä yksi audiolähde voi lähettää audiovirtaa useisiin audiovastaanottimiin, mutta yksi vastaanotin voi vastaanottaa audiovirtaa vain yhdeltä audiolähteeltä kerrallaan.

Sääntöjen priorisointi toimii siten, että audiolähteen liittyessä sääntöjä luetaan ensimmäiseen täsmäävään sääntöön asti ja toteutetaan sen säännön sisältämä toiminnallisuus.

RoutingRuleEngine ei tue tällä hetkellä päättelykoneelle tyypillistä ketjuttamista enempää kuin yhden ”lenkin” suuntaansa. Yleisesti ottaen päättelykoneet toimivat siten, että yhtä toimintoa voi seurata toinen toiminto, toista kolmas ja niin edelleen. Tähän syynä on tätä toiminnallisuutta vaativan käyttötapauksen puute sekä käytössä olleen ajan loppuminen.

RoutingRuleEnginen tekoäly on käytännössä sarja merkkijonon sisältöä tarkastelevia if-lauseita, tarkastelun perusteella suoritettavaa käsittelyä ja tämän käsittelyn perusteella saavutetun tiedon vertaamista tietopohjan sisältöön. RoutingRuleEnginellä ei ole omaa tietopohjaa, vaan se saa kutsun yhteydessä parametrinä tarvitsemansa osuuden RoutingServicen tietopohjasta. Esimerkiksi audiolähteen rekisteröityessä RoutingServiceen RoutingService kutsuu sääntökonetta antaen sille parametriksi juuri rekisteröityneen audiolähteen tiedot ja Mapin, joka sisältää kaikki tietopohjassa olevat audiovastaanottimet. Näiden parametrien avulla RoutingRuleEngine selvittää, täsmääkö annettu audiolähde yhteenkään sääntöön, ja jos se täsmää, RoutingRuleEngine etsii

tietopohjasta sääntöön täsmäävät audiovastaanottimet ja palauttaa niiden tiedot RoutingServicelle.

Kuviossa 28 on esitelty osa audiolähteen rekisteröitymisen yhteydessä kutsuttavasta funktiosta, joka tarkastaa, sopiiko tietopohjassa sijaitseva audiovastaanotin säännössä määriteltyihin parametreihin. Funktion avulla tarkastetaan jokainen tietopohjassa sijaitseva audiovastaanotin. Audiovastaanottimen täsmätessä sääntöön se lisätään palautettavaan listaan.

Kuviossa 28 on esitelty vain pieni osa käsittelystä. Todellisuudessa funktio on suhteellisen massiivinen, mikä johtuu tavasta, jolla ensin tarkastellaan, mitä operaattoreita säännöstä löytyvä merkkijono sisältää, sen jälkeen tarkastellaan mitä audiovastaanottimeen liittyviä parametrejä merkkijono sisältää, ja lopulta tarkastetaan parametrien yhteensopivuutta parametrinä saatujen vastaanottimen tietojen kanssa. Tuloksena on satoja rivejä raskaasti ylläpidettävää koodia. Tämän kaltainen ratkaisu oli jo RoutingRuleEnginen alkutaipaleella käytössä yksinkertaisen käyttötapauksen kokeilussa, ja sen toimiessa opinnäytetyön tekijä teki virhearvion toteuttaa loputkin käyttötapaukset tällä tavoin. Päätöksen seurauksena kului huomattavia määriä aikaa satoihin riveihin keskinkertaista koodia, kun ajan olisi voinut käyttää dynaamisempien ratkaisujen miettimiseen ja testailuun.

```
175  bool RoutingRuleEngine::checkSinkMatch(RoutingIFData::AudioSink sink,
176                                         QString ruleString, string appHost){
177      bool check = false;
178      ruleString = ruleString.simplified();
179      ruleString.replace( " ", "" );
180      QStringList sinkParams = getRuleParams("SinkParameters");
181      qDebug() << ruleString;
182
183      if (ruleString.contains("AND")){
184          if (ruleString.contains(sinkParams[0]) &&
185                  !ruleString.contains(sinkParams[1]) &&
186                  ruleString.contains(sinkParams[2])){ // sourceType & sourceHost
187              string sinkType;
188              string sinkHost;
189              QStringList pieces = ruleString.split("AND");
190
191              for (auto piece : pieces){
192                  QStringList pieces2 = piece.split("=");
193                  if (pieces2.value(0) == sinkParams[0]){
194                      sinkType = pieces2.value(1).QString::toStdString();
195                  }else if (pieces2.value(0) == sinkParams[2]){
196                      sinkHost = pieces2.value(1).QString::toStdString();
197                  }
198              }
199
200              if (sinkType == "*"){
201                  sinkType = sink.type;
202              }
203              if (sinkHost == "*"){
204                  sinkHost = appHost;
205              }
206
207              if (sink.type == sinkType && appHost == sinkHost){
208                  check = true;
209              }
210          }
```

Kuvio 28. Yksinkertaisen vain AND-operaattoria ja kahta parametriä sisältävän säännön käsittely

4.7 RoutingData

RoutingData-luokka toimii ohjauspalvelun tietopohjana. Tämä tarkoittaa, että luokan muuttujiin on säilötty kaikki RoutingServicen toimintaan ja ohjauksien päättelyyn tarvittava tieto siihen yhteydessä olevista sovelluksista. RoutingService-luokalla on pääsy tähän tietoon. Suunnitteluvaiheessa tietopohjan toteutukseen käytiin läpi muutamia vaihtoehtoja: säilytetäänkö data sovelluksen muuttujissa, jonkinlaisessa tiedostossa vai tietokannassa? Käytännössä oikea valinta määrittyy säilytettävän datan luonteen

ja määrän perusteella. Kysymyksiin, täytyykö datan säilyä ajokertojen välissä ja paljonko dataa on voitava säilyttää kerralla saa yleensä vastauksen perehtymällä vaatimusmäärittelyn käyttötapauksiin.

RoutingService palvelu perustuu Websocketin yli ylläpidettäviin yhteyksiin asiakkaiden ja RoutingServicen välillä. Käytännössä kaikki RoutingServicen säilyttämä tieto on siis ajonaikaista, joten tiedon ei tarvitse säilyä ajokertojen välissä. Edellä mainitusta syystä voidaan poissulkea tietokantaratkaisut ja tiedostoon tallentamisen ainakin ensimmäisen kysymyksen osalta. Säilytettävän tiedon määrän osalta tulee perehtyä vaatimusmäärittelyyn ja käyttötarkoitukseen. Opinnäytetyöntekijän ajatusprosessi kolmen tiedonsäilytysvaihtoehdon väliltä valittaessa on vetää häilyvät rajat eri tietuemäärien välille: Mikäli säilytettävänä on alle tuhat tietuetta tai joitain tuhansia tietueita, voidaan käyttää ohjelman sisäisiä muuttujia. Mikäli säilytettävänä on useita tuhansia tai kymmeniä tuhansia tietueita, voidaan tieto tallentaa tiedostoon. Mikäli tallennettavana on enemmän kuin kymmeniä tuhansia tietueita, käytetään tietokantaa. Nämä rajat ovat vain suuntaa antavia ja päätöstä tehtäessä tulee aina ottaa huomioon käyttötarkoitus ja säilytettävän tiedon luonne. Vaatimusmäärittely määrittelee yhden RoutingServicen säilytettäväksi tietuemääräksi joitain satoja ja mahdollisesti joitain tuhansia tietueita. Edellä mainituin perustein päätettiin toteuttaa tietopohja käyttämällä sovelluksen muuttujia.

Millainen muuttuja sopii RoutingServicen tietopohjan käyttötarkoitukseen? Säilytettävät tietueet ovat käytännössä olioiksi koottuja sovelluksia, audiolähteitä, audiovastaanottimia ja niiden välisiä audiovirtoja. Nämä oliot tulee myös yksilöidä duplikaattien välttämäseksi ja yksittäisen tietueen haun helpottamiseksi. Ensimmäisenä mieleen tulee säilyttää olioita listassa. Ohjelmistotekniikassa lista on abstrakti tietotyypi, joka voi sisältää laskettavissa olevan määrän arvoja, ja sama arvo voi esiintyä listassa useita kertoja. Vaatimusmäärittelyn mukaan kahta täysin samanlaista tietuetta ei esiinny käyttötapauksissa, joten listaa käytettäessä yksilöivä toiminnallisuus tulisi toteuttaa itse. Vaikka tämä olisikin mahdollista tarkistamalla aina ennen lisäystä, sisältääkö lista lisättävän tietueen duplikaattia, löytyy vielä muitakin vaihtoehtoja kuten Map.

Map eroaa listasta siten, että se sisältää avainarvopareja. Tämä tarkoittaa, että sama avainarvopari ei voi esiintyä mapissa useammin kuin yhdesti. Tämä mahdollistaa projektin käyttötapauksessa kahden muuten arvoiltaan samanlaisen audiolähteen yksilöinnin.

Projektin toteutuksessa käytettiin Qt:sta löytyvää QMap-luokkaa sekä C++:n omaa std::map-luokkaa. Vaikka kahden eri luokan käyttäminen kuulostaa hieman sekavalta ja turhan monimutkaiselta, sen voi perustella ETS-viestien luomilla rajoituksilla. ETS-järjestelmään on sisäänrakennettu erittäin näppärä Streamer-luokka, jonka avulla kaikista C++:n std-datatyypeistä voi luoda esimerkiksi rajapinnoissa käytettyjä XML-viestejä. ETS:n Streamer-luokka ei kuitenkaan tue Qt:n datatyyppejä. Qt:n QMap-luokan ollessa huomattavasti mukavampi käyttää käytettiin sitä toteutuksessa aina kun mahdollista. Käytännössä std:map-luokkaa käytetään vain viestien lähetyksen ja vastaanoton yhteydessä. QMap-luokasta kuten Qt:n luokista yleensäkin löytyy sisäänrakennettu funktio, joka muuttaa Qt:n luokan oliosta sitä vastaavan std-luokan olion.

Kuvio 29 vertailee QMap-luokan ja std::map-luokan käyttöä. Kuviosta näkyy kehittäjän kannalta ikävä ominaisuus std::map-luokassa, joka hyväksyy vain pair-luokan olioita tietueikseen. Alimmaisena osoitetaan QMap-luokan näppäryys sen muuttamisessa std::map olioksi.

```
45  void addSourceQt(){
46      QMap<string, RoutingAudioSource> sourceMap;
47      RoutingAudioSource routingSource();
48      sourceMap.insert(routingSource.uuid, routingSource);
49  }
50
51  void addSourceStd(){
52      std::map<string, RoutingAudioSource> sourceMap;
53      RoutingAudioSource routingSource();
54      sourceMap.insert(std::pair<string, RoutingIFData::AudioStream> (routingSource.uuid, routingSource));
55  }
56
57  std::map<string, RoutingAudioSource> fromQmapToStd(QMap<string, RoutingAudioSource> sourceMapQt){
58     std::map<string, RoutingAudioSource> sourceMapStd;
59     sourceMapStd = sourceMapQt.toStdMap();
60     return sourceMapStd;
61  }
```

Kuvio 29. Vertailu tietueen lisäyksestä std::map- ja QMap-luokkien välillllä sekä QMap-olion muuttaminen std::map-olioksi

RoutingData-luokka rakentuu neljästä jäsenmuuttujasta ja näiden muuttujien käsittelyyn liittyvistä funktioista. RoutingData-luokalla on QMap-tyyppinen jäsenmuuttuja jokaista RoutingServicen tarvitsemaa tietuetyyppiä kohden. Tietuetyypit ovat AudioSource, AudioSink, AudioStream ja ServiceProvider.

Kaikki ohjelmiston sovellukset käyttävät yhteisiä AudioSource-, AudioSink-, AudioStream- ja ServiceProvider-luokkia, jotka on määritelty kaikille sovelluksilla saatavilla olevassa nimiavaruudessa RoutingIFData:ssa. RoutingService tarvitsee enemmän tietoa siihen liittyneistä olioista kuin kaikille saatavissa olevissa luokissa on järkevää esittää. Tästä syystä RoutingData käyttää omia pääluokista perittyjä luokkia, joille on lisätty tarvittavia jäsenmuuttujia, kuten yhteyden tila tai vastaanottaako audiovastaanotin audiovirtaa vai ei.

4.8 Käyttöliittymä

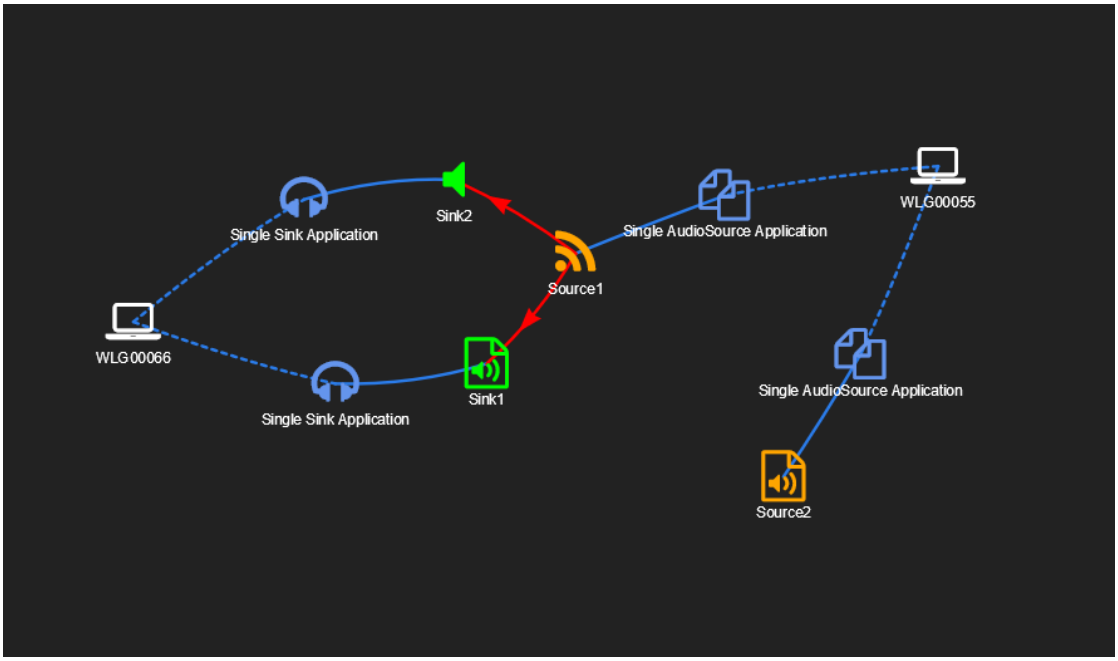
Käyttöliittymän kehittäminen palvelulle oli ylimääräinen tehtävä, joka toteutettaisiin vain, mikäli aikaa jäisi muiden toimintojen kehittämisen osalta. Onneksi tämä koski vain opinnäytetyön tekijää, ja muutaman viikon ajaksi projekti sai vahvistuksena erään toimeksiantajan kokeneen ohjelmoijan, Jani Honkosen. Käyttöliittymän toteutti Jani Honkonen opinnäytetyön tekijän toteuttaessa viestirajapinnan käyttöliittymän ja RoutingServicen väliseen kommunikointiin.

Käyttöliittymä toteutettiin vaatimusmäärittelyn mukaan web-pohjaiseksi sovellukseksi, jota käytettäisiin yleisimmillä web-selaimilla. Palvelinpää on toteutettu Node.js:llä ja se otetaan käyttöön käynnistämällä palvelin komentoriviltä. Node.js palvelin ilmoittaa käynnistyksen yhteydessä isäntäkoneensa hostnamen, johon yhdistetään web-selaimella.

Web-selaimeen ilmestyvän näkymän avulla voi ohjata audiovirtoja vastaanottimille tai poistaa olemassa olevia ohjauksia. Tämän näkymän visualisointi on toteutettu Vis.js-nimisellä kirjastolla, joka tarjoaa valmiita ratkaisua erinäköisten solmukohtien välisten yhteyksien visualisoinnille. Tässä tapauksessa tälläisia solmuja (engl. node) on kolmen tasoisia: Host, Service ja Application. Käytännössä yhdellä hostilla voi olla useita servicejä ja yhdellä servicellä useita applikaatioita. Eri applikaatioiden välillä voi olla striimejä.

Myös käyttöliittymä käyttää hyväkseen ETS:n viestiprotokollaa viestien lähettämiseen ja vastaanottamiseen, mutta huomioitavaa on, että XML:n sijasta käyttöliittymä esittää viestit JSON-muodossa.

Käyttöliittymä tarjoaa toiminnallisuuden audiovirtojen ohjaamiseen ja näiden ohjausten poistamiseen manuaalisesti. Tulevaisuudessa toiminnallisuuteen lisätään mahdollisesti applikaatioiden ja servicejen poistaminen sekä sääntöjen määrittäminen. Kuviossa 30 kuvataan erilaisia solmukohtia ja niitä yhdistäviä viivoja. Oranssilla värillä kuvatut solmut ovat audiolähteitä ja vihreällä kuvatut solmut ovat audiovastaanottimia.



Kuvio 30. Kuvakaappaus käyttöliittymän näkymästä

4.9 Testaus

Työn testaus toteutettiin käytännössä ainoastaan manuaalisesti työn tueksi luotujen testaussovellusten avulla. Tämän kaltainen testaus jättää paljon toivomisen varaa, sillä sovelluksen kehittäjän testatessa omaa tuotostaan muodostuu ainakin kaksi ongelmaa. Sovelluksen kehittäjä vastaa yleensä huonosti loppukäyttäjää, sillä sovelluksen kehittäjällä on tapana käyttää sovellusta täsmälleen siten, kuin hän on sitä tarkoittanut käytettävän. Tästä syystä jää helposti huomaamatta erinäisiä käyttöön tai käyttöliittymään liittyviä puutteita.

Toteutuksen päättelykone taasen voi sisältää huomattavia määriä erilaisia parametrien ja operaattoreiden yhdistelmiä, joista näin ollen suurinta osaa ei ole ikinä testattu toiminnassa.

Ensimmäisen ongelma on hankala ratkaistava sovelluksen kehittäjälle. Toki kokenut kehittäjä voi osata huomioida erinäköiset käyttöön liittyvät mahdollisuudet ja näin saada karsittua puutteita pois. Paras ratkaisu on tuoda projektiryhmään testaaja, joka luultavasti luonnostaan huomaa käyttöön liittyvät puutteet testaamisen ohella. Toisen ongelman ratkaisuun taas voi löytyä ratkaisu myös kehittäjältä itseltään. Sovelluksen kehittäjä voisi luoda sääntöihin liitetyt yksikkötestit, joiden avulla voitaisiin automatisoida erilaisten sääntöjen toimivuus tietyissä tilanteissa.

Testauksen keveys johtuu myös projektin tutkivasta luonteesta. Toimeksiantajalta oli suhteellisen suuri sijoitus tukea tutkimusprojektia kahdella kokeneella työntekijällä, joista kumpaakaan ei ollut tarkoitettu testaamaan projektissa toteutettuja sovelluksia. Opinnäytetyön tekijän keskittyminen testaukseen olisi saattanut vähentää saadun tutkimustiedon määrää ja oikean testaajan budjetoiminen tutkimusprojektiin olisi käynyt liian kalliiksi, joten testauksen osuus rajattiin käytännössä pois.

5 Tulokset

Toteutusvaiheen tuloksena syntyi kokonaisuutena toimiva ohjelmisto, johon kuuluvat sovellukset pystyvät kommunikoimaan toistensa kanssa toteutettujen rajapintojen avulla, sekä sovellus, joka pystyy päättelemään suoritettavia toimintoja ennalta määritettyjen sääntöjen perusteella.

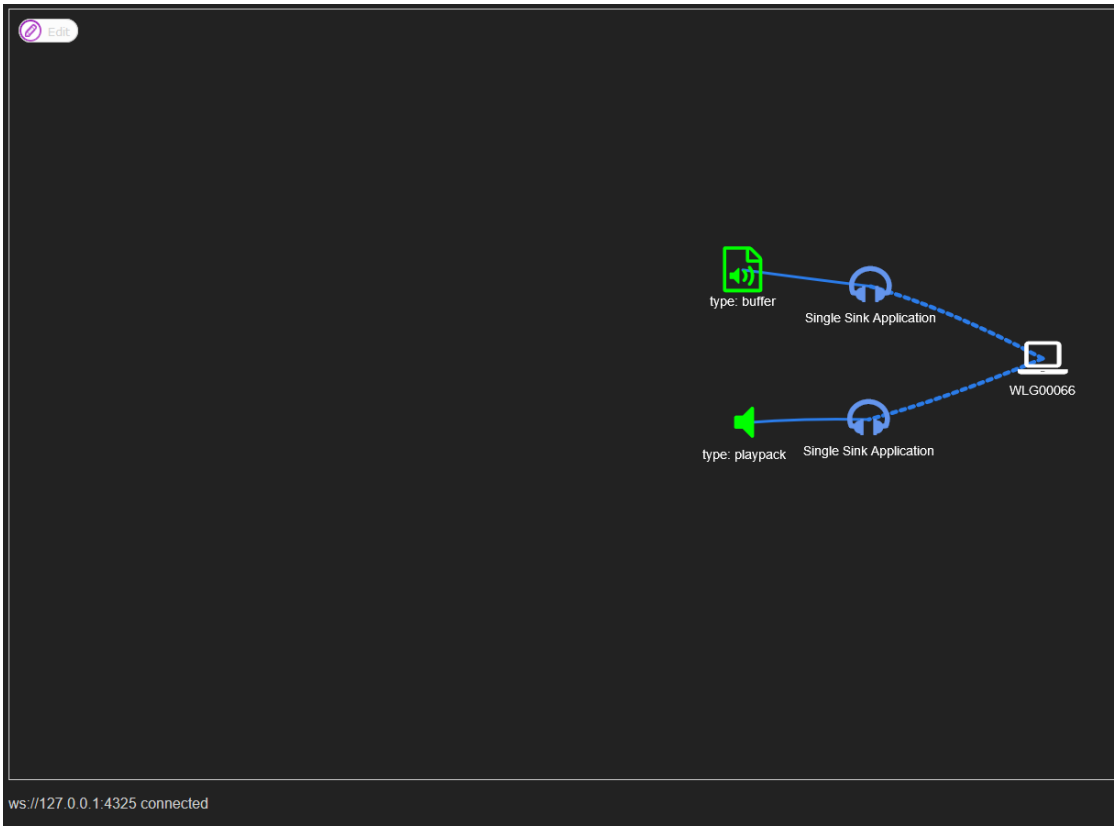
Toiminnallisuuden havainnollistamiseksi otetaan esimerkki käyttötapaus: Käyttäjänä minulla on AudioChannel ja Buffer-tyyppisiä audiolähteitä sekä Playback- ja Buffer-tyyppisiä audiovastaanottimia. Käyttäjänä haluan, että isäntäkoneesta WLG00055 yhdistävät AudioChannel tyyppiset audiolähteet ohjataan lähettämään audiovirtaa isäntäkoneesta WLG00066 löytyviin Playback-tyyppisiin vastaanottimiin. Tämän lisäksi käyttäjänä haluan, että isäntäkoneesta WLG00055 yhdistävät Buffer-tyyppiset audiolähteet ohjataan lähettämään audiovirtaa isäntäkoneesta WLG00066 löytyviin Buffer-tyyppisiin vastaanottimiin.

Tämän kaltaisen käyttötapauksen toteutus onnistuu kahdella kuviossa 31 esitetyllä säännöllä.

```
<Rule>
    <Name>Rule1</Name>
    <Source>sourceType=audioChannel AND sourceHost=WLG00055</Source>
    <Sink>sinkType=playback AND sinkHost=WLG00066</Sink>
</Rule>
<Rule>
    <Name>Rule2</Name>
    <Source>sourceType=buffer AND sourceHost=WLG00055</Source>
    <Sink>sinkType=buffer AND sinkHost=WLG00066</Sink>
</Rule>
```

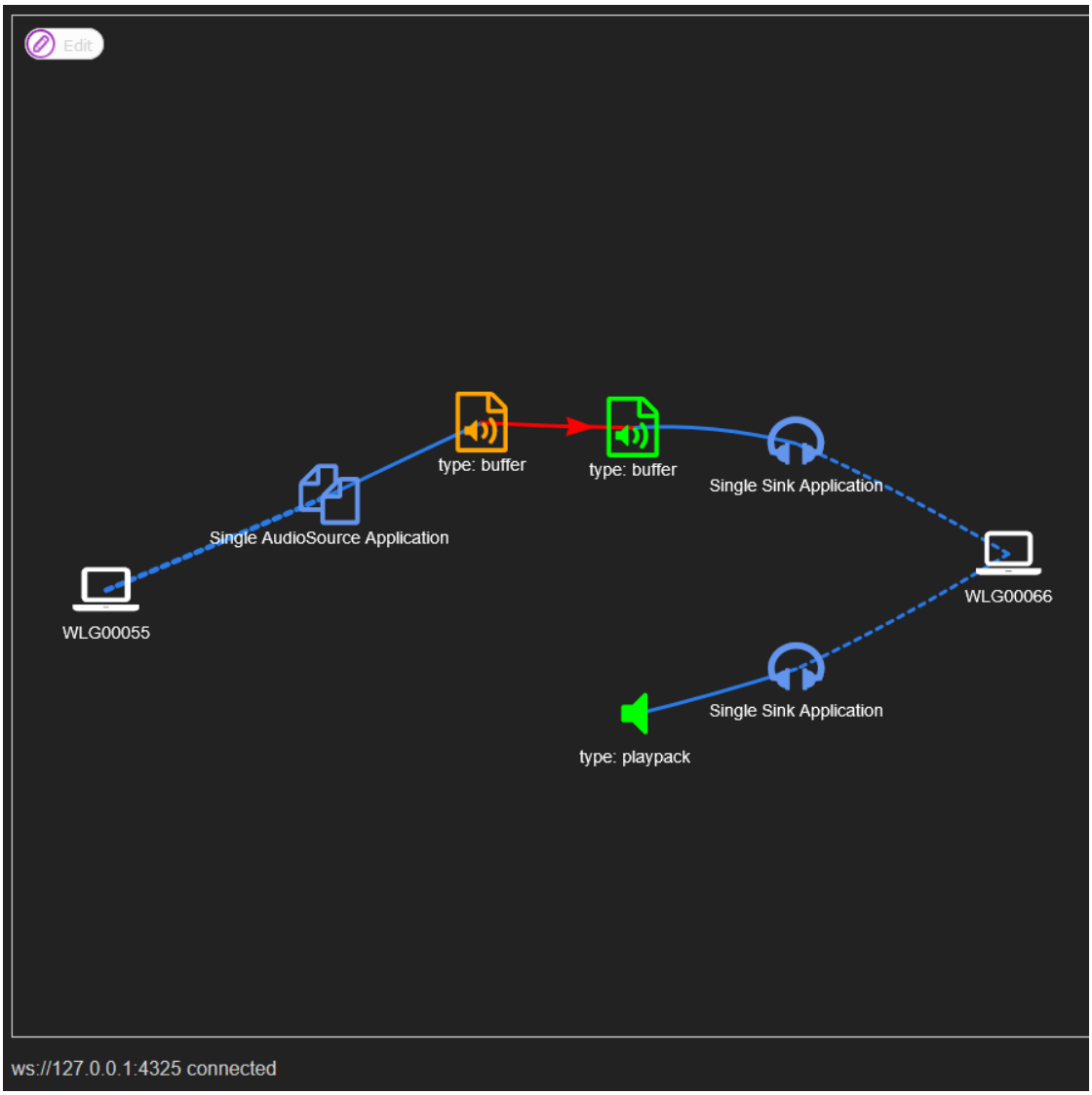
Kuvio 31. Edellä mainitun käyttötapauksen toteuttavat säännöt

Kuvio 32 kuvaa tilannetta, jossa RoutingServicen tietopohjasta löytyy kaksi yhdistänyttä audiovastaanotin-tyypin sovellusta. RoutingRuleEnginelle on määritetty kaksi edellä mainittua sääntöä.



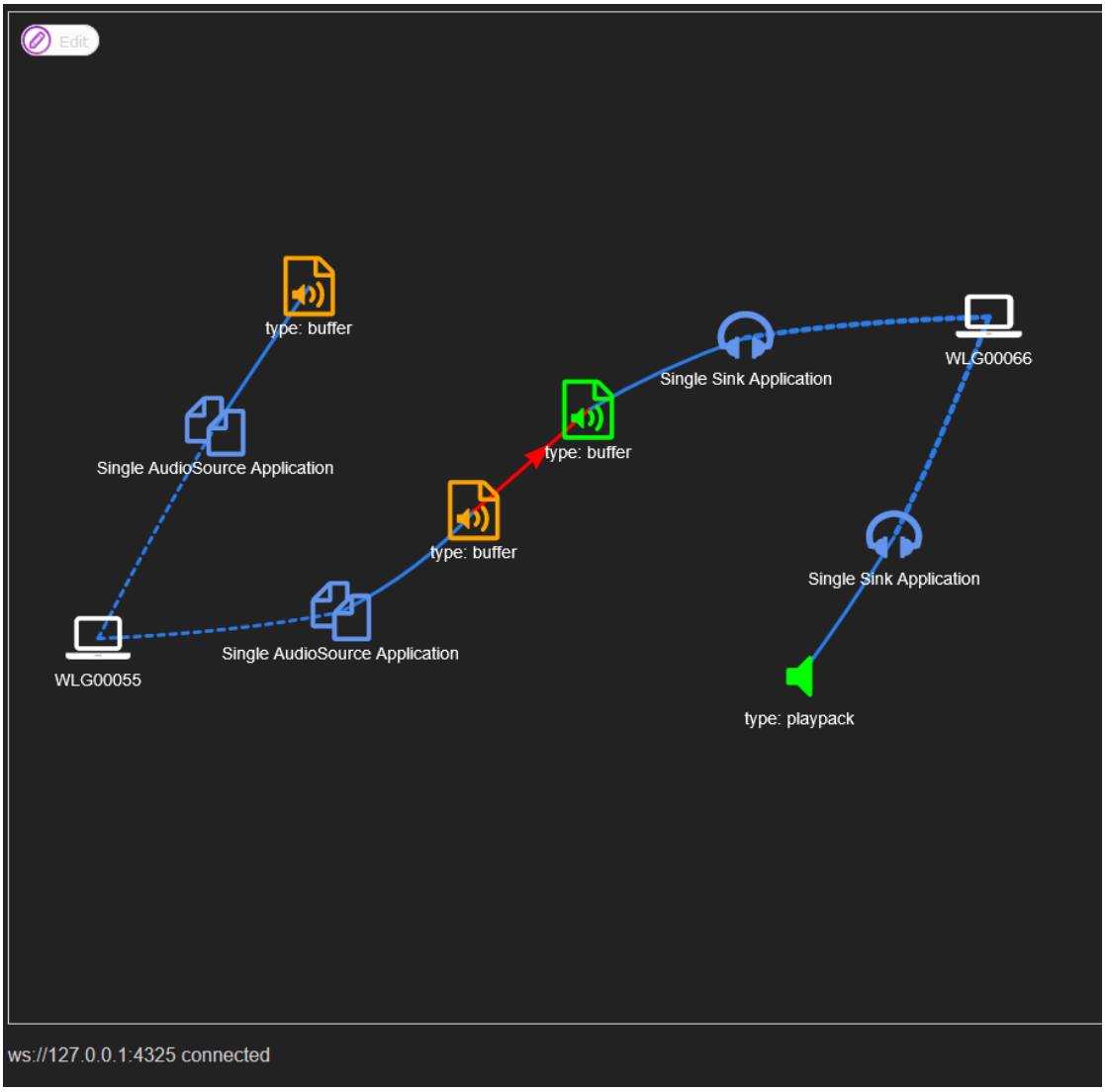
Kuvio 32. Alkutilanne, jossa kaksi audiovastaanotin-tyypin sovellusta on liittynyt isäntäkoneesta WLG00066

Kuvio 33 kuvaa tilanteen, jossa RoutingServiceen yhdistää audiolähde-tyypin sovellus. RoutingService kutsuu RoutingRuleEngineä, joka tarkistaa, täsmääkö yhdistänyt audiolähde yhteenkään sääntöön. Huomatessaan audiolähteen täsmäävän Rule2:een se käy läpi tietopohjasta löytyvät audiovastaanottimet ja tarkastaa, täsmääkö niistä yksikään Rule2:een. Löytäessään täsmäävän vastaanottimen se palauttaa RoutingServicelle sen tiedot, jotka RoutingService kertoo yhdistäneelle audiolähteelle ja ohjaa sen lähettämään audiovirtaa audiovastaanottimeen.



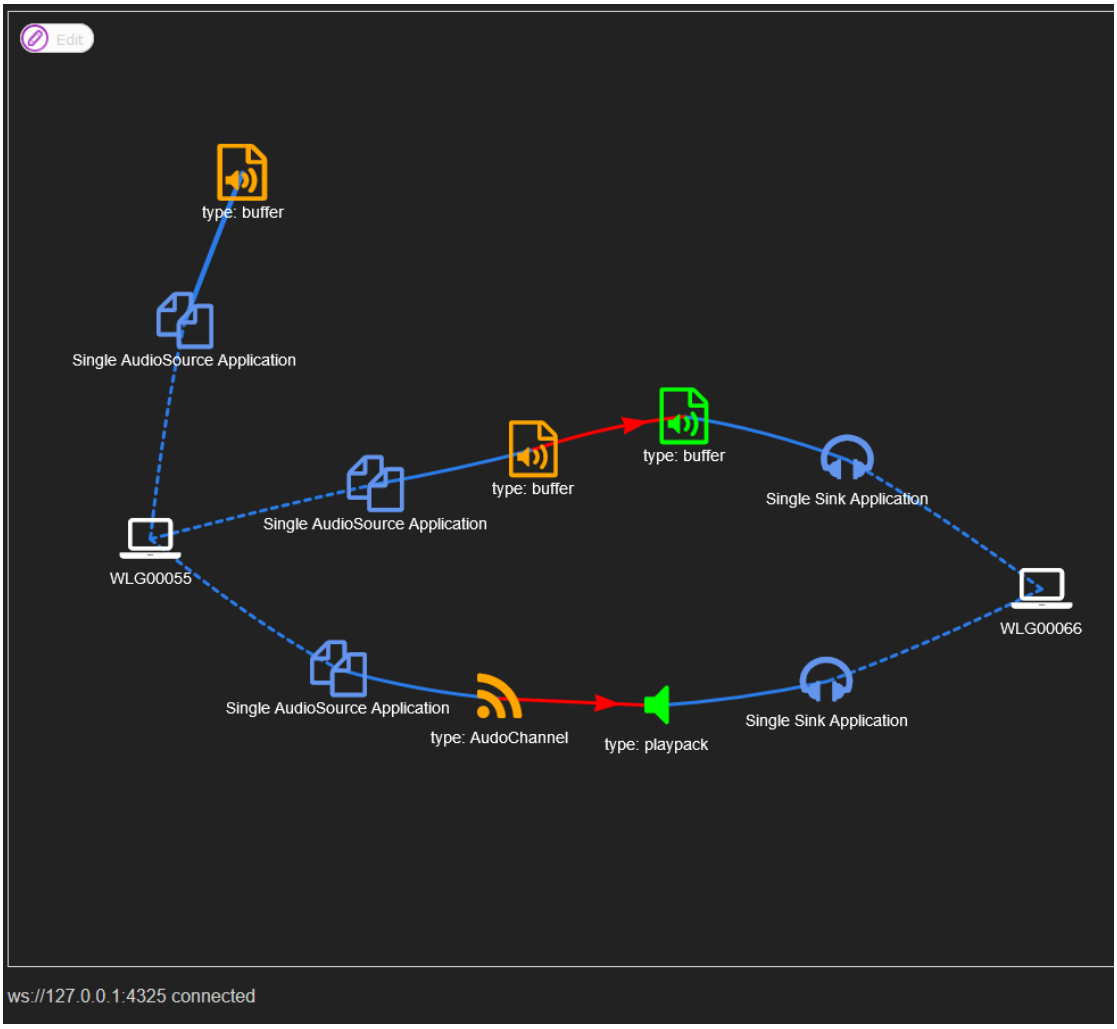
Kuvio 33. Tilanne, jossa hostista WLG00055 liittyy source-tyypin sovellus

Kuviossa 34 toinen buffer-tyyppinen audiolähde yhdistää palveluun, mutta täsmäävän audiovastaanottimen puuttuessa audiolähde jää odottamaan sopivan audiovastaanottimen yhdistämistä audiovirran lähettämiseksi.



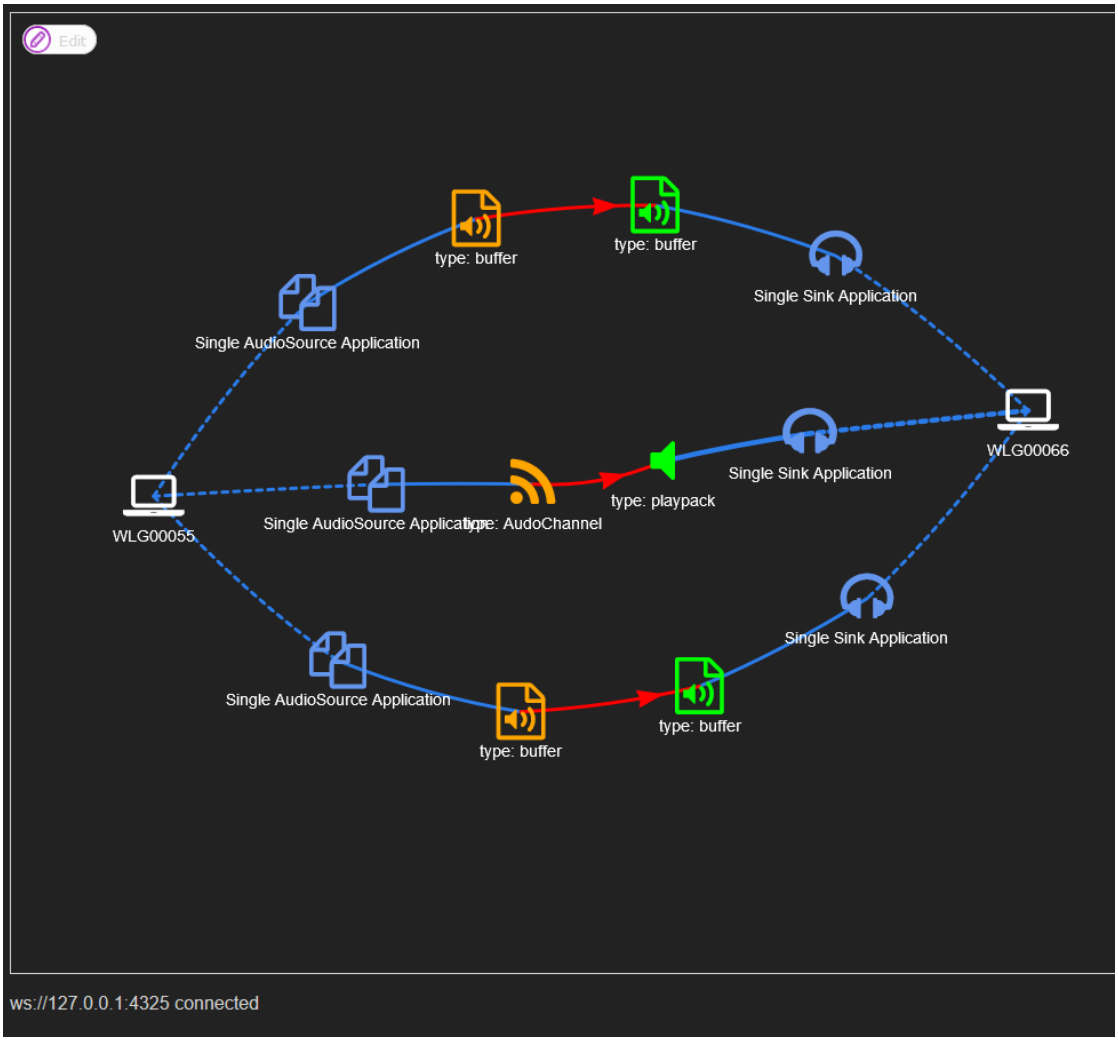
Kuvio 34. Toinen buffer-tyypin audiolähde yhdistää palveluun

Kuviossa 35 käydään läpi täsmälleen samat askeleet kuin Kuviossa 33, mutta täsmäävä sääntö on vain eri.



Kuvio 35. AudioChannel-tyyppinen audiolähde yhdistää palveluun

Kuviossa 36 uusi buffer-tyyppinen vastaanotin yhdistää RoutingServiceen ja sen täsmätessä sääntöön RoutingRuleEngine tarkistaa, löytyykö tietopohjasta samaan sääntöön täsmäävää audiolähdettä. Audiolähteen löytyessä RoutingService ohjaa aiemmin yhdistäneen buffer-tyyppisen audiolähteen lähettämään audiovirtaa juuri yhdistäneeseen vastaanottimeen.



Kuvio 36. Buffer-tyyppinen audiovastaanotin yhdistää palveluun

Tämän käyttötapauksen ollessa melko yksinkertainen RoutingServicen ja RoutingRuleEnginen avulla pystyy määrittelemään myös monimutkaisempia käyttötapauksia. Esimerkiksi NOT-operaattorin hyödyntäminen tapauksessa, jossa tiettyyn isäntäkoneeseen halutaan hyväksyä kaiken muun tyyppiset audiovirrat yhtä lukuunottamatta, tai OR-operaattorin käyttö tapauksessa, jossa hyväksytää audiovirrat tasan kolmesta eri isäntäkoneesta.

6 Johtopäätökset

6.1 Arvio toteutuksesta

Työn toteutuksessa opinnäytetyön tekijän vastuut voi jakaa kolmeen suurempaan komponenttiin. Prioriteettijärjestyksessä ne olivat toimivat ja tarkoituksenmukaiset ohjausrajapinnat sovellusten väliseen kommunikointiin, audiovirtojen ohjauskesta vastaava palvelu, joka voi päätellä käyttäjän puolesta, miten audiovirrat ohjataan ennalta määritettyjen sääntöjen perusteella, ja käyttöliittymä, jonka avulla audiovirtoja voi ohjata manuaalisesti.

Vaikka työn tutkimuksen pääpainona on ehdottomasti automatisoinnin mahdollistaminen, rajapintojen toteutus voitti priorisoinnissa, sillä toimivat rajapinnat olivat välttämättömät automatisoinnin kehityksessä ja testauksessa.

Ajallisesti projekti toteutettiin noin kahdeksan työviikon eli noin 300 tunnin aikana. Tästä ajasta noin puolet käytettiin rajapintojen suunnitteluun, toteutukseen ja paranteluun, kun taas toinen puolikas ohjauspalvelun ja päättelykoneen suunnitteluun, toteutukseen ja paranteluun. Projektin loppupuolella huomattiin, että käyttöliittymän toteutukselle ei jäisi aikaa, joten käyttöliittymän toteutuksesta vastasi Jani Honkonen, joka työskenteli toimeksiantajalta jo löytyvän sovelluksen parissa.

Rajapintojen osuus työstä kuulostaa turhan suurelta, mutta niiden suunnittelussa otettiin huomioon myös RoutingService-sovelluksen toiminta, joten niiden kehitystyö liittyy vahvemmin RoutingService-sovellukseen, kuin aluksi saattaisi ajatella. Rajapintoihin käytetty aika näkyy myös lopputuloksessa, sillä ne ovat toimiva osa kokonaisuutta ja niiden käyttäminen on suhteellisen helppoa.

Ohjauspalvelu on rajapintojen tavoin toimiva osa kokonaisuutta. Myös kapseloinnissa onnistuttiin, sillä ohjauspalvelu ja päättelykone ovat kaksi erilistä komponenttia, joita voi käyttää myös ilman toisiaan.

Toteutuksen keskeneräisimmäksi komponentiksi jäi audiovirtojen ohjauksen automatisointiin käytettävä päättelykone. Vaikka päättelykone tämän hetkisessä tilassa on toimiva osa kokonaisuutta, siinä on tiettyjä puutteita. Päättelykone pystyy tulkitsemaan säännöistä loogisilla operaattoreita AND, OR, NOT eroteltuja parametrejä ja

päättelemään niiden avulla käyttäjän haluamaa toiminnallisuutta. Nykyisessä tilassaan tämä kattaa kuitenkin vain melko yksinkertaisia sääntöjä. Tuomalla esimerkiksi sulkumerkit ja lisää jokerimerkkejä sääntöjen määrittelyyn olisi mahdollista kattaa kaikki mahdolliset käyttötapaukset. Toki erilaisia käyttötapauksia on huomattavia määriä, mutta siitä huolimatta ne tulisi pyrkiä kattamaan.

Päättelykone ei ole myöskään koodillisesti aivan tyydyttävällä tasolla. Tämän hetken tilassa se osaa käsitellä kolmea eri parametriä jokaista audiolähdettä tai vastaanotinta kohden. Parametrit ovat nimi, tyyppi ja hostname. Jos loppukäyttäjä haluaisikin käyttää jotain neljättä parametria, päättelykoneen koodia pitäisi päivittää. Tämän toiminnallisuuden olisi suotavaa olla täysin dynaamista, mutta toisaalta sovellus on toteutettu parissa kuukaudessa käytännössä yhden junior developerin voimin, joten se on prototyyppiasteelle tyydyttävä toteutus.

Projektin tavoitteena oli tutkia ongelmakenttää ja toteuttaa prototyyppijärjestelmä. Vaikka jatkokehityksen tarvetta jäi, opinnäytetyöntekijän mielestä tavoitteet täytettiin.

6.2 Arvio työskentelystä

Työskentelystä suurin osa tapahtui toimeksiantajan tiloissa, toimeksiantajan laitteilla. Vaikka tuotos on ainakin vielä vain yrityksen sisäisen prototyyppijärjestelmä ilman oikeaa asiakasta, oli työnteossa kuitenkin tiettyä aitouden tuntua. Projektiryhmässä mukana olleet toimeksiantajan työntekijät olivat aidosti kiinnostuneita suunnittelusta, kehittämisestä sekä tuloksista. Etenkin suunnitteluun apua sai runsaasti. Usein näin muutaman vuoden ohjelmointikokemuksella ideoita saattaa olla paljonkin, mutta usein puuttuu tietynlainen itseluottamus ideoiden oikeellisuuden suhteen. Varmuuden idean oikeellisuudesta tai palautteen sen puutteista sai työskennellessä helposti vain kysymällä mielipidettä alalla 10–15 vuotta työskennelleeltä ammattilaiselta. Tämän kaltaisen avun arvoa ei voi liikaa korostaa.

Opinnäytetyöntekijä oli aidosti kiinnostunut aiheesta, mikä varmasti selittää korkean työmotivaation. Motivaation piti korkealla myös onnistunut projektinhallinta. Scrummallin helppous ja toimivuus tulivat todistettua projektin toteutuksen aikana sen helpottaessa työn jaksottamista ja opettaessa työmäärän arviointia.

Työskentelyn ollessa sujuvaa taustatyön olisi voinut hoitaa hieman paremmin. Vaikka toimeksiantaja antoi aiheen hyvissä ajoin ennen toteutusvaiheen alkua, ymmärrys aihealueeseen kasvoi eniten projektin edetessä, ja joitain hyviä selkeitä lähteitä löytyi vasta projektin loppupuolella. Mikäli taustatyö olisi onnistunut paremmin, toteutusvaihe olisi saattanut olla helpompi ja nopeampi.

6.3 Jatkokehitysideat

Lopullisen toteutuksen ollessa tyydyttävä myös jatkokehitettävää jäi runsaasti.

Ensimmäisenä jatkokehityksen alkaessa paranneltaisiin päättelykoneen koodia. Koodin tulisi olla dynaamisempaa käyttäjän määrittelemien parametrien suhteen ja sen tulisi ottaa huomioon esimerkiksi sulkumerkit sääntöjen määrittelyssä.

Kuvion 37 mukaisen säännön tulkinta tulisi olla mahdollinen. Nykyisestä toteutuksesta puuttuu sulkumerkit ja ”?” jokerimerkki.

```
<Rule>
    <Name>Rule1</Name>
    <Source>sourceType=buffer? AND sourceHost=*</Source>
    <Sink>(sinkType=playback AND sinkHost=WLG00055) NOT sinkHost=WLF?</Sink>
</Rule>
```

Kuvio 37. Jatkokehityksessä mahdollistettava sääntö

Päättelykoneen toteuttavan luokan nimi on edelleen sen alkutaipaleelta RoutingRuleEngine (suom. sääntökone). Sääntökone-termi kuvaa ainoastaan eteenpäin ketjuttavia toteutuksia. RoutingRuleEnginen toiminnallisuuteen kuuluu myös taaksepäin ketjutus mikä tekee siitä tarkalleen ottaen päättelykoneen (engl. inference engine), joten luokan nimi tulisi muuttaa kuvaavammaksi.

RoutingServicen eri olioiden välistä toiminnallisuutta tulisi muuttaa hyödyntämään Qt:n näppärää Signals & Slots-menetelmää. Tämä olisi ollut toteutusvaiheen yksi seuraavista muutoksista, mikäli aikaa olisi ollut enemmän käytössä.

Virheenkäsittely toimii käytännössä if-else lauseilla, joissa suoritetava toiminto on if-lauseen sisällä, ja sen epäonnistuessa else-lauseen sisältä tulostetaan debug dataa

konsoliin epäonnistumisen syistä. Tämänkaltainen virheenkäsittely nopeuttaa kehitystyötä, mutta ei ole tarpeeksi luotettava julkaistavassa sovelluksessa. Jatkokehityksessä virheenkäsittely korvattaisiin try catch-rakenteella.

Käyttöliittymä näyttää hyvältä ja toimii tarkoituksenmukaisesti. Tarkoituksella useita sovelluksia ja audiovirtoja sisältäväksi simuloitu näkymä kuitenkin osoittaa sen heikkouden. Liitteessä 1 on kuvankaappaus käyttöliittymästä, joka sisältää muutamia isäntäkoneita ja kymmeniä audiolähteitä sekä vastaanottimia. Jo tuo näkymä vaikuttaa melko sekavalta ja käyttötapauksen mukaan on mahdollista olla satoja tai jopa tuhansia RoutingServiceen liittyneitä audiolähteitä tai vastaanottimia. Käyttöliittymää tulisi jatkokehittää jonkinlaisella ylimääräisiä tai turhia solmuja piilottavalla logiikalla.

6.4 Johtopäätökset

Opinnäytetyö kokonaisuudessaan oli äärimmäisen opettava kokemus. Käytännössä prosessiin kuului työnhakua aiheen saamiseksi, työyhteisöön tutustumista, suunnittelua, projektityöryhmän osana työskentelyä, kokeneemmilta oppimista, omien aikaansaannosten demoamista, epäonnistumisia ja onnistumisia sekä tietysti dokumentointia.

Työuran aloittaminen oikean projektin toteutusta mukailevalla tavalla oli myös miellyttävä kokemus. Toiminnon huolellinen suunnittelu, sen toimimaan saaminen ja lopulta demossa esittely on erittäin palkitsevaa. Etenkin toimintojen suunnittelu, suunnitelman läpikäynti työryhmän kanssa ja saadun palautteen pohjalta suunnitelman muuttaminen oli mielekästä toimintaa. Yrityksen sisäisestä kehitysprojektista saadun kokemuksen ja itseluottamuksen avulla on myös helpompi lähteä työskentelemään oikeaan asiakasprojektiin.

Toteutettavat toiminnallisuudet olivat toimeksiantajan puolesta selkeästi määritelty. Rajapintojen toteutuksen ollessa olennaista RoutingServicen toiminnalle niiden toteuttaminen vei turhan paljon aikaa pois päättelykoneen kehitykseltä jättäen sen hieman keskeneräiseltä vaikuttavaan, joskin toimivaan tilaan. Toisaalta keskeneräinenkin toteutus tuotti toimeksiantajalle arvokasta tietoa tämän kaltaisen järjestelmän kehityksen vaikeusasteesta ja mahdollisista kustannuksista.

Alussa asetetut tavoitteet henkilökohtaisesta kehittymisestä, toimeksiantajalle tuotetusta tutkimustiedosta ja prototyypin toteutuksesta täyttyivät, vaikka jatkokehitykselle jääkin huomattavasti tilaa. Projektiryhmän kesken toteutettu ohjelmisto on toimiva kokonaisuus, joka sisältää myös mahdollisille asiakkaille näytettävää miellyttävän näköistä esimerkkiä toiminnastaan. Kaiken kaikkiaan projektin voi katsoa onnistuneen.

Lähteet

About Qt. 2017. Qt kehitysympäristön wikisivut. Muokattu 14.2.2017. Viitattu 28.6.2017. http://wiki.qt.io/About_Qt

Agile software development. 2017. Wikipedia-artikkeli. Muokattu 16.8.2017. Viitattu 21.8.2017. https://en.wikipedia.org/wiki/Agile_software_development

Ampache. N.d. Ampache streaming serverin kotisivut. Viitattu 28.3.2017. http://ampache.org/

Application Programming Interface. 2017. Wikipedia-artikkeli. Muokattu 24.6.2017. Viitattu 28.6.2017 https://en.wikipedia.org/wiki/Application_programming_interface

Audio signal. 2017. Wikipedia-artikkeli. Muokattu 20.6.2017. Viitattu 22.8.2017. https://en.wikipedia.org/wiki/Audio_signal

Britton, C. 2008. Choosin a Programming Language. Viitattu 12.8.2017. https://msdn.microsoft.com/en-us/library/cc168615.aspx

Combitech. N.d. Combitech Oy:n kotisivut. Viitattu 1.7.217. http://combitech.fi

Communications protocol. 2017. Wikipedia-artikkeli. Muokattu 31.7.2017. Viitattu 3.8.2017 https://en.wikipedia.org/wiki/Communications_protocol

Data transmission. 2017. Wikipedia-artikkeli. Muokattu 30.7.2017. Viitattu 3.8.2017 https://en.wikipedia.org/wiki/Data_transmission

ETS Framework. N.d. Combitech Oy:n ei-julkinen dokumentaatio ETS-järjestelmästä.

Expert System. 2017. Wikipedia-artikkeli. Muokattu 8.5.2017. Viitattu 1.7.2017. https://en.wikipedia.org/wiki/Expert_system

Getting Started – Git Basics. N.d. Git versionhallintajärjstelmän kotisivut. Viitattu 21.8.2017. https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Git Branching. N.d. Git versionhallintajärjstelmän kotisivut. Viitattu 21.8.2017. https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Gitlab. 2017. Wikipedia-artikkeli. Muokattu 19.8.2017. Viitattu 21.8.2017. https://en.wikipedia.org/wiki/GitLab

Haikala, I & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Hämeenlinna: Kariston Kirjapaino Oy

History of C++. N.d. C++-ohjelmointikielen kotisivut. Viitattu 28.6.2017. http://www.cplusplus.com/info/history/

Icecast FAQ. N.d. Icecast-sovelluksen kotisivut. Viitattu 28.3.2017. http://icecast.org/faq/

Inference engine. 2017. Wikipedia-artikkeli. Muokattu 23.6.2017. Viitattu 28.6.2017 https://en.wikipedia.org/wiki/Inference_engine

Internet. 2017. Wikipedia-artikkeli. Muokattu 23.7.2017. Viitattu 3.8.2017 https://en.wikipedia.org/wiki/Internet

Internet Protocol Suite. 2017. Wikipedia-artikkeli. Muokattu 3.8.2017. Viitattu 3.8.2017 https://en.wikipedia.org/wiki/Internet_protocol_suite

Introducing JSON. N.d. JSON-tekstiformaatin kotisivut. Viitattu 28.6.2017 http://json.org/

Java (Programming Language). 2017. Wikipedia-artikkeli. Muokattu. 19.6.2017. Viitattu 12.8.2017. https://en.wikipedia.org/wiki/Java_(programming_language)

JavaScript. 2017. Wikipedia-artikkeli. Muokattu 18.8.2017. Viitattu 21.8.2017. https://en.wikipedia.org/wiki/JavaScript

Jira (software). 2017. Wikipedia-artikkeli. Muokattu 6.8.2017. Viitattu 21.8.2017. https://en.wikipedia.org/wiki/Jira_(software)

Järvinen, P. 2003. IT-tietosanakirja. Porvoo: WS Bookwell.

Markup language. 2017. Wikipedia-artikkeli Muokattu 19.8.2017. Viitattu 21.8. 2017. https://en.wikipedia.org/wiki/Markup_language

Node.js. 2017. Wikipedia-artikkeli. Muokattu 27.6.2017. Viitattu 28.6.2017 https://en.wikipedia.org/wiki/Node.js

Ohjelmistotuotanto. 2016. Wikipedia-artikkeli. Muokattu 6.1.2016. Viitattu 25.7.2017 https://fi.wikipedia.org/wiki/Ohjelmistotuotanto

Pulse-code modulation. 2017. Wikipedia-artikkeli. Muokattu 3.8.2017. Viitattu 23.8.2017. https://en.wikipedia.org/wiki/Pulse-code_modulation

Redmine. N.d. Redmine projektinhallintajärjestelmän kotisivut. Viitattu 21.8.2017. http://www.redmine.org/

Routing. 2017. Wikipedia-artikkeli. Muokattu 28.6.2017. Viitattu 15.8.2017. https://en.wikipedia.org/wiki/Routing

Signals & Slots. N.d. Qt kehitysympäristön dokumentaatiosivut. Viitattu 26.7.2017 http://doc.qt.io/qt-4.8/signalsandslots.html

Taina, J. 2010. Ohjelmistojen vaatimusmäärittely. Opetusmateriaali. Helsingin yliopisto, Tietojenkäsittelytieteen laitos. Viitattu 1.7.2017. https://www.cs.helsinki.fi/u/taina/ovm/k-2010/pdf/kalvot1-42_6.pdf

Transport Layer Security. 2017. Wikipedia-artikkeli. Muokattu 3.8.2017. Viitattu 3.8.2017 https://en.wikipedia.org/wiki/Transport_Layer_Security

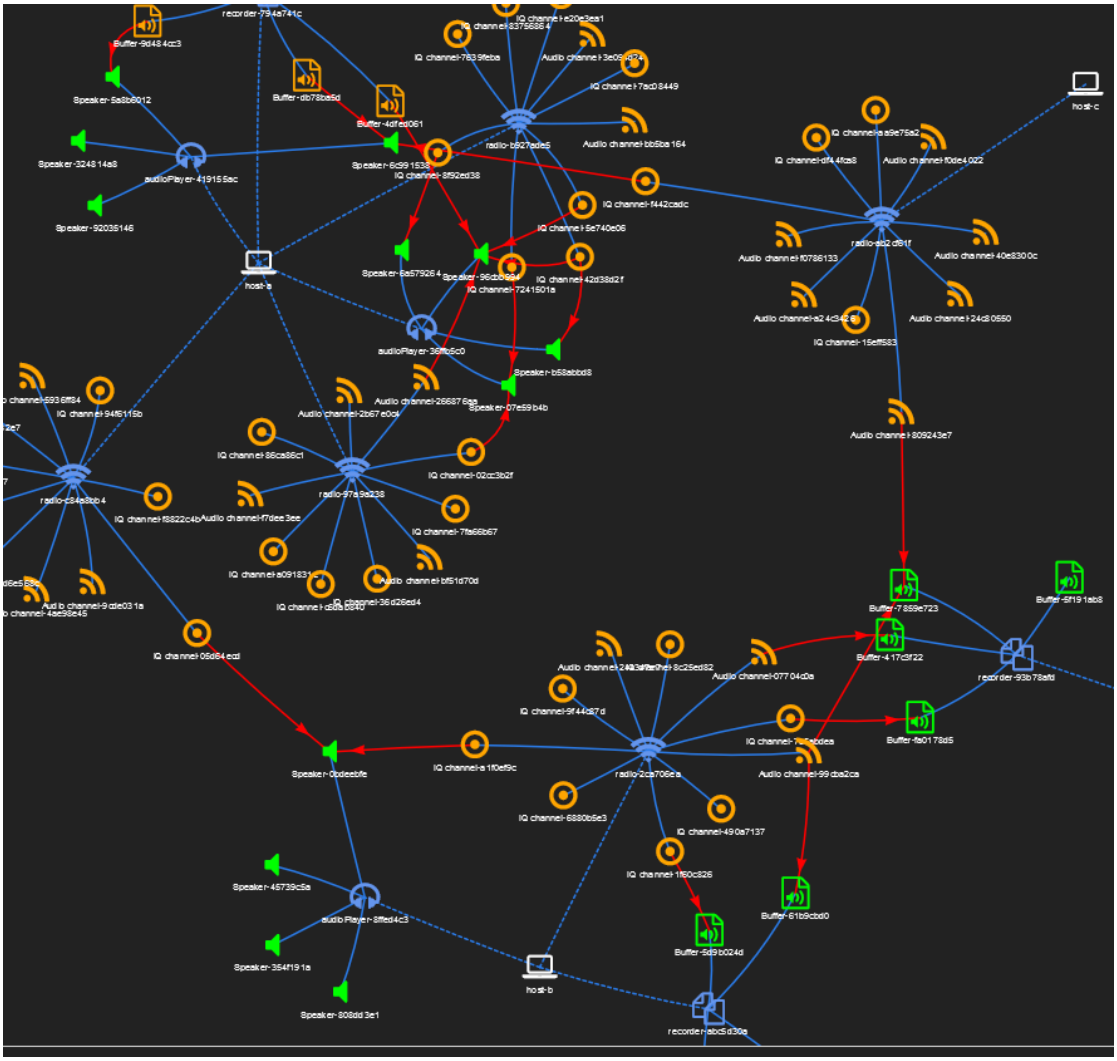
Using the Meta-Object Compiler (moc). 2016. Qt-kehitysympäristön dokumentaatiosivut. Viitattu 28.6.2017 http://doc.qt.io/qt-4.8/moc.html

vis.js. 2017. vis.js-kirjaston kotisivut. Viitattu 28.6.2017 http://visjs.org/

XML. 2017. Wikipedia-artikkeli. Muokattu 27.6.2017. Viitattu 28.6.2017 https://en.wikipedia.org/wiki/XML

Liitteet

Liite 1. Kuvakaappaus simuloidusta käyttöliittymänäkymästä



Liite 2. Ohjelmiston käyttämät abstraktit tietotyypit